

PREPAR3D

Scripting Overview

Overview

Prepar3D supports two different scripting languages:

- [Reverse Polish Notation \(RPN\) Scripting](#)
- [Lua Scripting](#)

Reverse Polish Notation (RPN) Scripting

[Gauges](#) and [Scenario Scripting](#) support expressions in the form of Reverse Polish Notation (RPN) which is a mathematical notation where the operators follow all of the operands.

See the [RPN Scripting](#) article to read about how to access variables in this notation as well as construct expressions.

Lua Scripting

[Gauges](#) and [Scenario Scripting](#) support expressions in the form of Lua Script which is a light-weight scripting language that supports multi-paradigm programming.

See the [Lua Scripting](#) article to read about how to access variables in this language as well write full fledged scripts.

CONTENTS

- [RPN Scripting](#)
- [Lua Scripting](#)

RELATED LINKS

- [Scenario Scripting](#)
- [Programmable Gauges](#)

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

RPN Scripting

Overview

In addition to the newer [LUA Scripting](#), Prepar3D supports Reverse Polish Notation (RPN) scripting. The evaluation of RPN scripts is done using a stack and reverse polish notation. For example, if you want a gauge to show altitude in thousands of feet, divide the value of INDICATED ALTITUDE by 1000. In RPN this would be:

```
(A:INDICATD ALTITUDE, feet) 1000 /
```

To execute an arithmetic operation, put both operands first in the script and then include the symbol of operation (the divide symbol in this case). This is how some programmable calculators performed arithmetic operations in the past.

This convention works the following way: When the script parser comes across a value, it pushes it onto the top of the stack. When the script parser comes across an operator, it pops from the stack the number of operands that the operator works on (usually one or two values).

Whatever value is left on top of the stack at the end of the execution is the result of the calculated expression. The [Examples](#) section provides a list of simple and more complex script examples.

The [Variable Overview](#) article describes all of the systems that support variable access.

CONTENTS

- [Operators](#)
- [Examples](#)
- [The Infix2Postfix Tool](#)

RELATED LINKS

- [Scripting Overview](#)
- [Variable Overview](#)
- [Creating XML Gauges](#)
- [Scenario Scripting](#)
- [SDK Overview](#)

Operators

The operators that are available in RPN are as follows:

Operator	Operation	Arguments	Example	Result
Common Operator				
+	addition	2	3 5 +	8
-	subtraction. If the stack contains A B - , then the calculation is A - B .	2	(L:Value) 90 -	The local value minus 90.
/	division. If the stack contains A B / , then the calculation is A / B .	2	5 2 /	2.5
*	multiplication	2	pi 2 *	2 pi
%	taking modulo	2	5 3 %	2
++	increment	1	4 ++	5
--	decrement	1	4 --	3
/-/ neg	negates a number	1	4 /-/	-4
Comparison Operators				
==	true if equal	2	(L:Value) 0 == if{ A }	Operation A is carried out if Value is 0.
!=	true if not equal	2	(L:Value) 0 != if{ A }	Operation A is carried out if Value is not 0.
>	true if greater than	2	(L:Value1) (L:Value2) > if{ A } else{ B }	If Value1 is greater than Value2, operation A is carried out, otherwise operation B is carried out.
<	true if less	2	(L:Value1) (L:Value2) < if{ A } else{ B }	If Value1 is less than Value2, operation A is carried out, otherwise

			if <i>B</i>	operation B is carried out.
>=	true greater than or equal	2	(L:Value1) (L:Value2) >= if{ A } else{ B }	If Value1 is greater than or equal to Value2, operation A is carried out, otherwise operation B is carried out.
<=	true if less than or equal	2	(L:Value1) (L:Value2) <= if{ A } else{ B }	If Value1 is less than or equal to Value2, operation A is carried out, otherwise operation B is carried out.
?	The third operand determines whether the first (True) or second (False) is selected.	3	A B True ?	This evaluates to A.
Bit Operators				
&	bitwise AND	2	5 3 &	1
 	bitwise OR	2	5 3 	7
^	bitwise XOR	2	5 3 ^	6
~	bitwise NOT	1	5 ~	-6
>>	shift right operand number of bits	2	5 3 >>	0
<<	shift left operand number of bits	2	5 3 <<	40
Logical Operators				
!, NOT	NOT	1	(L:Local) ! (>L:Local)	Toggles the variable Local
&&, AND	AND	2	(L:Local) 0xFF00 && (>L:Local)	The variable Local is ANDed with hex 0xFF00
 , OR	OR	2	(L:Local) 07777 OR (>L:Local)	The variable Local is ORed with octal 7777.
Numerical Operators				
abs	Absolute value	1	-5 abs	5
int flr	Calculates nearest integer number which is less than the source number	1	5.98 flr	5
rng	Range; returns True if the third operand lies between values one and two.	3	4 7 6 rng	True
cos	Cosine (input in radians)	1	pi cos	-1
lg	Logarithm to base 10	1	10 lg	1
min	Minimum	2	5 2 min	2
sin	Sine (input in radians)	1	pi sin	0
acos	Arc cosine (returns radians)	1	pi acos	
ctg	cotangent (input in radians)	1	pi ctg	
ln	Natural logarithm	1	2.718282 ln	1
sqr	Square	1	5 sqr	25
asin	arc sine	1	pi asin	
eps	Floating-point relative accuracy	1	1 eps	$2^{(-52)}$
log	Logarithm of operand one, to the base of operand two.	2	8 2 log	3

pi	Pi = 3.14159; puts pi on the stack	0	pi	3.14159
sqrt	Square root	1	25 sqrt	5
atg2	arc tangent with two inputs (input in radians)	2		
exp	Exponent; e to the power of the operand	1	1 exp	2.718282
max	Maximum	2	5 2 max	5
pow	Power of; the first value to the power of the second	2	2 5 pow	32
tg	Tangent (input in radians)	1	pi tg	0
atg	arc tangent with one input	1	pi atg	
Special Operators				
div	Divides integers; its result is always an integer	2	5 3 div	1
ceil	Calculates nearest integer number which is bigger than the source	1	4.3 ceil	5
near	Calculates the nearest integer number, rounding .5 up.	1	4.5 near	5
dnor d360 rdeg	Normalizes an angle expressed in degrees. The result is a value between 0 and 360.	1	-15 dnor	345
rddg	Converts radians to degrees	1	pi rddg	180
dgrd	Converts degrees to radians	1	180 dgrd	pi
rnor	Normalizes an angle expressed in radians, the result of this operation is between 0 and 2 pi	1	5 pi	1.8584
if{ }	If statement, note there is no space between the if and the {	1	(L:Value) 0 == if{ A }	Operation A is carried out if Value is 0.
els{ }	Else statement, note there is no space between the els and the {	1	(L:Value1) (L:Value2) <= if{ A } els{ B }	If Value1 is less than or equal to Value2, operation A is carried out, otherwise operation B is carried out.
quit	The quit statement allows expression evaluation to stop completely, and avoid the use of nesting if{ statements.	0	pi quit (L:Value1) (L:Value2) <= if{ A } els{ B }	pi. The rest of the script is ignored.
g0...gn	Goto label. Execution will jump to the specified label. Labels are set by entering a colon followed by the label number.	0	g4	Execution jump to :4
			50 40 30 20 10 E	The "5" indicates there are five case values, which are selected depending on the evaluation of (L:value). If the evaluation is

case	Case statement		Lv 3 (L:value) case	equal to or greater than 0, but less than 1, the result is 10. If the evaluation is equal to or greater than 1, but less than 2, the result is 20, and so on.
String Operators				
lc	Converts a string to lowercase	1	'ABcd10' lc	'abcd10'
uc cap	Converts a string to uppercase	1	'ABcd10' uc	'ABCD10'
chr	Converts a number to a symbol	1	65 chr	'A'
ord	Converts a symbol to an integer	1	'A' ord	65
scat	Concatenates strings	2	'abc' 'red' scat	'abcred'
schr	Finds a symbol in a string			
scmp	Compares strings, case sensitive	2	(M:Event) 'LeftSingle' scmp 0 == if{ A }else{ B }	Performs A if the left mouse button has been pressed, otherwise performs B
scmi	Compares strings, ignoring case	2	'left' 'Left' scmi 0 == if{ 'yes' }	'yes'
sstr	Finds a substring	2	'cd' 'abcde' sstr	2
ssub	Extracts a substring	2	'ab' 'abcde' ssub	'cde'
symb	Extracts a single character	2	'abc' 1 symb	'b'
Stack Operators				
b	Backup the stack	0		
c	Clears the stack	0	stack: 1 2 3 c	stack:
d	Duplicates the value that is on the top of the stack	1	stack: 5 d	stack: 5 5
p	Pops and discards the top value on the stack	1	stack: 1 2 3 p	stack: 1 2
r	Reverses the top and second values on the stack	2	stack: 1 2 3 r	stack: 1 3 2
s0, s1, ... s49	Stores the top value in an internal register, but does not pop it from the stack.	1	stack: 1 2 3 s0	stack: 1 2 3 s0: 3
l0, l1, ... l49	Loads a value from a register to the top of the stack	1	stack: 1 2 3 s0 l0	stack: 1 2 3 3
sp0, sp1, ... sp49	Stores the top value and pops it from the stack	1	stack: 1 2 3 sp0	stack: 1 2 sp0: 3

Notes

- Formatted strings use a similar but slightly different syntax, see the [example](#).
- Strings should be entered using single quotes, for example: 'abcd'.
- Hexadecimal numbers can be entered using the 0xf or 0xF convention (for example, 0xff or 0xFF00FF00)
- Octal numbers can be entered by using a leading zero. For example, 022 is octal 18. Be careful not to enter leading zeros on decimal numbers.
- Scientific notation can be used: 5E2 represents 5×10^2 (500), 5E-2 is 0.005

Examples

The table below gives a range of script examples that can be used for aircraft. See also the examples for the [GaugeText](#) object, and a few examples in the [Gauge Color Scripts](#) section.

Aircraft Script Examples

#	Expression	Gauge	Description
1	<code>(A:LIGHT NAV, bool)</code>	Navigation Lights	Returns True (1) if the navigation light switch is on, False (0) otherwise.
2	<code>(A:TRAILING EDGE FLAPS LEFT ANGLE, radians) 1.1 *</code>	Flaps	Returns the left flaps angle in radians, multiplied by 1.1. Multiplications like this can help reduce rounding errors when making comparisons with integer settings.
3	<code>(A:PARTIAL PANEL ELECTRICAL,enum) !</code>	Electrical gauges	The enum for this variable is (0 = OK, 1= fail, 2 = blank), so in this case the result is inverted by the "!" operator to give 1 = OK and 0 = fail.
4	<code>(A:PLANE HEADING DEGREES GYRO, degrees) /- dgrd</code>	HSI Compass ring	Returns the aircraft heading negated and converted to radians.
5	<code>(A:NAV GSI:1,percent) 250 /</code>	HSI Glideslope deflection	Returns the percentage for the GSI (Glideslope deviation indicator) for Nav1, divided by 250.
6	<code>(P:Units of measure, enum) 2 == if{ (A:RADIO HEIGHT, meters) } els{ (RADIO HEIGHT, feet) }</code>	Radio Height (or Radar Altitude) needle	The Units of measure enum is: 0 = English 1 = Metric (with altitude in feet) 2 = Metric (with altitude in meters) So if this enum value is 2, the If statement is evaluated, otherwise the Els statement is evaluated.
7	<code>(A:PARTIAL PANEL ELECTRICAL,enum)! if{ 25 (A:GENERAL ENG FUEL PRESSURE:1, psi) - } els{ 25.5 }</code>	Fuel Pressure	If the Partial Panel Electrical is OK (see Example 3 above) then 25 minus the fuel reading for engine 1 is returned, if not the fixed value of 25.5 is returned.
8	<code>(A:FUEL TANK SELECTOR 1, enum) 0 == (A:FUEL TANK SELECTOR 1, enum) 3 == or if { pi 2 / } els{ 0 }</code>	Right Valve of Fuel Gauge	If all the fuel tanks are selected (FUEL TANK SELECTOR enum value is 0) or the right fuel tank is selected (the enum value is 3) then the fuel valve is set at pi/2, otherwise it is set at zero.
9	<code>(A:NAV1 OBS, degrees) d (A:PARTIAL PANEL HEADING, bool) (A:PARTIAL PANEL ELECTRICAL, bool) or 0 == if{ (A:PLANE HEADING DEGREES GYRO, degrees) 90 - - } dgrd</code>	GPS element of HSI	If the PARTIAL PANEL HEADING is false and the PARTIAL PANEL ELECTRICAL is false, then this expression returns the NAV1 OBS reading minus the (PLANE HEADING DEGREES GYRO reading minus 90), converted to radians.
10	<code>(P:Units of measure, enum) 2 == if{ (A:INDICATED ALTITUDE, meters) } els{ (A:INDICATED ALTITUDE, feet) } 100000 / 360 * dgrd</code>	Altimeter ten thousand foot needle	Divides the altitude by one hundred thousand, then multiplies by 360, and converts the result from degrees to radians.
11	<code>(L>Show Volts 1,bool) if{ (A:ELECTRICAL GENALT BUS VOLTAGE:1, volts) 2 * } els{ (A:ELECTRICAL GENALT BUS AMPS:1, amps) }</code>	Amps/Volts	If the local variable Show Volts 1 is true, the bus voltage for engine 1, multiplied by 2, is returned. Otherwise the bus amps for engine 1 is returned.
12	<code>(A:ADF Radial:1,degrees) 360 + d360 dgrd</code>	ADF	The ADF radial value in degrees is added to 360 and then normalized to a value between 0 and 360. It is then converted to radians.

Notes

- Simulation variables are not case-sensitive, they can be all upper, all lower or a mix of upper and lower case in your scripts.

String Formatting

Certain XML Gauge Elements, such as the [GaugeText Objects](#), and Scenario Objects, such as the [On Screen Text](#), support using scripting to more intricately format the text that is to be displayed. While [GaugeText Objects](#) support the full list of options to follow, [On Screen Text](#) only supports the text based formatting options.

NOTE: This syntax is **NOT** the same as generic RPN script syntax and care must be taken not to confuse the two.

Formatting Numbers

The format for numbers is contained within the !...! marks.

The last letter is required and is case-sensitive, is the formatting of the variable, where:

- s = string.
- d = decimal number (integer). If the number is not an integer, it is rounded to the nearest integer. Note that rounding, not truncation occurs.
- f = number (floating point)

The formatting letter can be preceded by a number, which is the minimum number of digits to display, and is optional. For decimal numbers the following rules apply:

- If d is preceded by the digit "0", then leading zeros are added if necessary.
- If d is preceded by "+", text is left-aligned.
- If d is preceded by "+", a "+" symbol is indicated in front of the number when the number is greater than 0 (a "-" is always used to indicate numbers less than 0).
- If d is preceded by " " (space), leading spaces are added if necessary.

For floating point numbers, the following rule applies:

- If a decimal point is used in the formatting number, the digit after the decimal point specifies the number of digits to display after the decimal point.

Examples of Number Formatting

String	Result	Description
<code>%(` 12.34)%!4.3f!</code>	12.340	The 4 in 4.3 is ignored.
<code>%(` 12.34)%!04.3f!</code>	12.340	Leading "0"s are not added to floating point numbers.
<code>%(` 12345.6789)%!4.3f!</code>	12345.679	The number before decimal point does not limit the number of digits displayed before decimal point.
<code>%(` 34.56)%!+d!</code>	+35	Rounding, not truncation, has occurred.
<code>%(`(234)%!5d!</code>	234	Two leading spaces have been prefixed to the number.
<code>%(`('foo')%!5s!</code>	foo	Two leading spaces have been prefixed to the string
<code>%(`(234)%!3s!</code>	234	The number is output as a string, with a minimum of three digits.

Conditional Formatting

The format of conditions (if, then, else, and case statements) in gauge strings is different from that in other scripts. In gauge strings use the `%{if}`, `%{else}`, and `%{end}` constructs to choose which text to display. Note that these keywords are case-sensitive and must be typed in lowercase. Also, there must not be a space between the "%" and the "{". An if statement can be used without a corresponding else, in which case nothing is displayed if the result of the condition is false. The syntax for usage is one of the following:

- `%(`(CONDITIONAL)%{if}TEXT TO DISPLAY IF TRUE%{else}TEXT TO DISPLAY IF FALSE%{end}`
- `%(`(CONDITIONAL)%{if}TEXT TO DISPLAY IF TRUE%{end}`

For example: `%(`(1)%{if}ON%{else}OFF%{end}` would give the output ON. Whereas `%(`(0)%{if}The value is true%{else}The value is false%{end}` would give the output :The value is false.

In the [Example of Gauge Strings](#) below, note also the case and loop statements.

Escape Code Formatting

It is also possible to insert escape code sequences into gauge strings.

Escape code	Translation
<code>\{tabs=50R,60C,244L}</code>	Set 3 tab stops; the first is right-aligned, the second is centered, and last is left-aligned.
<code>\{fnt1}</code>	Switch to the first alternate font specified as a child of the gauge text element
<code>\{fnt}</code>	Return to the default font
<code>\{up}</code>	Superscript

\{dn\}	Subscript
\{md\}	Normal (neither superscript nor subscript)
\{bo\}	Bold
\{ul\}	Underline
\{itl\}	Italic
\{strk\}	Strikeout
\{blink\}	Blink
\{rev\}	Reverse background/foreground color for text
\{nr\}	Normal -- clear all properties previously set.
\{lcl\}	Line color
\{blc\}	Background line color
\{clr\}	Color
\{bck\}	Background color
\{dplo=X\}	Put a degrees symbol above the next character after the ?=?
\{dpl=XY\}	Make X superscript and Y subscript
\{lsp=23\}	Set line spacing to 23
\{lsp\}	Set line spacing to default
\{ladj=L\}	Set horizontal text alignment to left. (use ?C? for center or ?R? for right)
\{line=240\}	Draw a horizontal line with width 240
\{lmrg=20\}	Set the left margin to 20
\{rmrg=30\}	Set the right margin to 30
\{img1\}	Insert image #1 (a text element can have image children)

Examples

Gauge string	Description
Fuel Pressure	The text will appear exactly as entered: Fuel Pressure
Fuel Capacity: %((A:FUEL TOTAL CAPACITY))%!1.2f!	The fuel capacity of the aircraft will be given as a floating point number accurate to two decimal places, following the initial string, such as: Fuel Capacity: 80.55
%((A:ENG ON FIRE:1) (A:ENG ON FIRE:2) !if{ 'Warning: Engine Fire' })	The text string " Warning: Engine Fire " will appear if either or both of the engines are on fire.
%({ 1)%{if}ON%{else}OFF%{end}	The text ON would be rendered. If there is no {else} statement, then no text will be displayed if the condition evaluates to false.
%({ 3)%{case}%{ :0 }AIRPORT%{ :1 }INTERSECTION%{ :2 }NDB%{ :3 }VOR%{ :4 }MARKER%{end}	A case statement can be used to select a text string from a group of strings. The case numbers do not have to be sequential. The example would produce the result: VOR
%((C:Mission:OnScreenTimerValue) 60 / 60 / flr)%!02d! %((C:Mission:OnScreenTimerValue) 60 / flr 60 %)!%02d! %((C:Mission:OnScreenTimerValue) flr 60 %)!02d!. %((C:Mission:OnScreenTimerValue) 10 * flr 10 %)!%01d!	Takes the custom on-screen timer value and displays the time in hours, minutes, seconds and tenths of a second. The !02d! indicates that the text output should be displayed with two digits (for example, 06). The % signs inside the string refer to the modulus operator, and the colons and period between the statements will appear on the screen as text, for example: 01 : 14: 08 . 2
85 %%	To output the percent character, use two percent signs in the script: 85 %
%({10 s2 1 s1)%{loop}%({ l1)%!s! %({ l1 ++ s1 l2 <)%{next}	This statement sets up two registers s1 and s2, with the numbers 1 and 10, then loops to print out: 1 2 3 4 5 6 7 8 9 Whatever value is on top of the stack when the %({next) statement is reached is evaluated as a boolean to determine if execution of the loop should continue.

\b Indented Text	The parser will strip leading whitespace. A backspace '\b' can be used to override this behavior if indented text is required.
------------------	--

The Infix2Postfix Tool

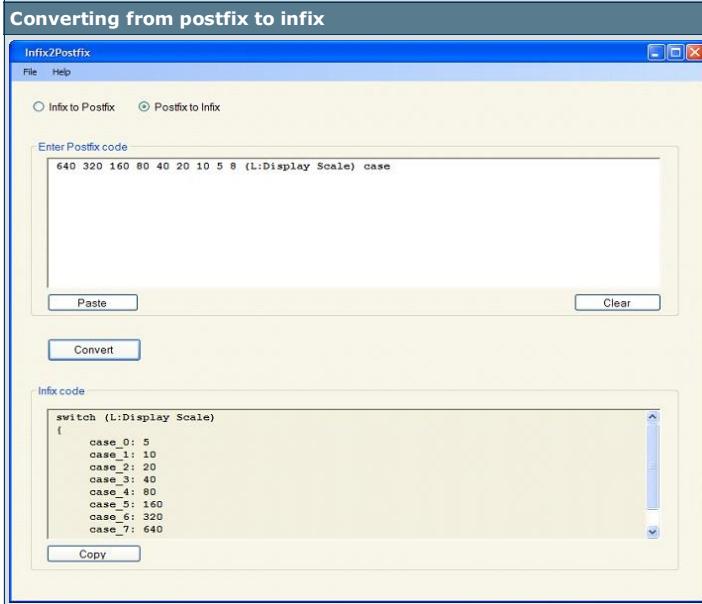
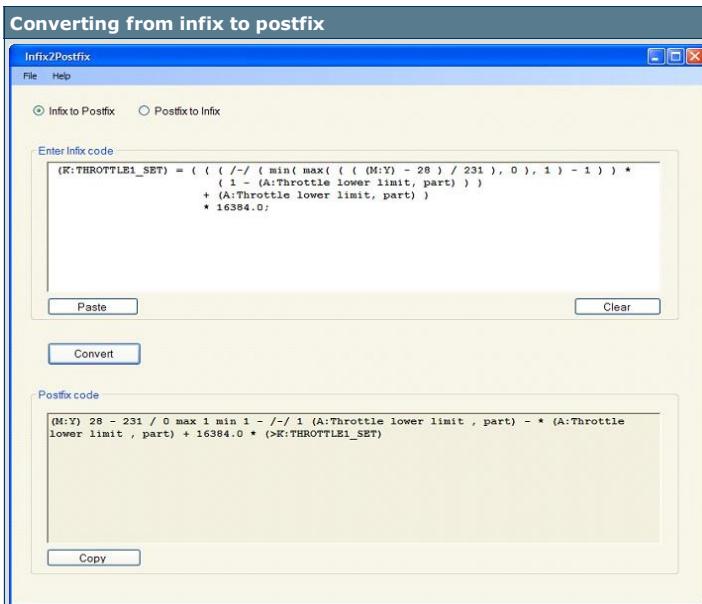
The Infix2Postfix tool makes it easier to both understand the logic of existing postfix (often called reverse Polish) notation, and enables the writing of scripts in the more normal infix notation familiar to all programmers. The Infix2Postfix tool is in the following folder:

SDK/Panels Gauges User Interface/Panels

The following screen shots show a conversion from infix to postfix, and vice versa. You can use the *Paste* button to take infix notation from the clipboard, and the *Copy* button to copy the postfix notation to the clipboard, so that it can be pasted into a gauge script. Alternatively use the *File* menu to select an existing XML gauge file, and you can then step through the scripts in the file, and examine the conversion to infix. The syntax for the infix notation is similar to C# or C++, and is available in a text file through an option in the *Help* menu.

Note

This tool is for educational purposes only, there are some scripts that it will not convert correctly.



[- top -](#)

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

Lua Scripting

Overview

In addition to the legacy [Reverse Polish Notation \(RPN\)](#) script, Prepar3D offers the capability to embed Lua scripts into xml gauges. Prepar3D currently uses Lua 5.3.3. For more specific information regarding this version of Lua please refer to the [Reference Manual](#).

In addition to embedding Lua scripts into XML gauges, scenarios can now use [Script Actions](#) and [Script Triggers](#) which can be scripted in Lua.

Syntax

The first line of the embedded Lua script must precede with **!lua**. The script itself can span across several lines. A comparison between getting the time in seconds for the second hand of a clock in standard gauge XML and Lua XML is shown below:

Standard XML syntax:

```
<Value>(P:Local time,seconds) flr 60 %</Value>
```

Lua XML Syntax

```
<Value>!lua return flr(varget("P:Local time", "seconds")) % 60
</Value>
```

Be aware that since the Lua code is embedded in XML escape characters are required instead of standard Lua syntax for the following characters:

- " " (when inside a string definition)
- ' ' (when inside a string definition)
- < <
- > >
- & &

Functions

In addition to the standard operations provided by Lua these commands are also available. The following table provides the Lua commands that are available.

Lua Function	Description	Example
lg(x)	log10(x)	<value>!lua return lg(1000)</value>
pi	3.141592653589793238462643	<value>!lua return pi()</value>
tg(x)	tan(x) (in radians)	<value>!lua return tg(45)</value>
abs(x)	abs(x)	<value>!lua return abs(-1)</value>
atg(x)	atan(x) (in radians)	<value>!lua return atg(1)</value>
cos(x)	cos(x) (in radians)	<value>!lua return cos(60)</value>
ctg(x)	cot(x) (in radians)	<value>!lua return ctg(45)</value>
eps(x,y,z)	epsilon (floating point equality within value z)	<value>!lua return eps(1.234, 1.234, 0.00001)</value>
exp(x)	e^x	<value>!lua return exp(2)</value>
flr(x)	floor(x)	<value>!lua return flr(1.2)</value>
int(x)	int cast	<value>!lua x = int(true)</value>
bool(x)	bool cast	<value>!lua x = bool(0)</value>
log(x,y)	log _y (x)	<value>!lua return log(10,2)</value>
max(x,y)	max(x,y)	<value>!lua return max(10,15)</value>
min(x,y)	min(x,y)	<value>!lua return min(10,15)</value>
pow(x,y)	x^y	<value>!lua return pow(2,3)</value>
sin(x)	sin(x) (in radians)	<value>!lua return sin(30)</value>
sqr(x)	x^2	<value>!lua return sqr(3)</value>
acos(x)	acos(x) (in radians)	<value>!lua return acos(0)</value>
asin(x)	asin(x) (in radians)	<value>!lua return asin(1)</value>
atg2(x,y)	atan2(x,y) (in radians)	<value>!lua return atan2(1,0)</value>

CONTENTS

- [Syntax](#)
- [Functions](#)
- [Examples](#)

RELATED LINKS

- [Scripting Overview](#)
- [Variable Overview](#)
- [Creating XML Gauges](#)
- [Scenario Scripting](#)
- [SDK Overview](#)

ceil(x)	ceil(x) (in radians)	<value>!lua return ceil(1.2)</value>
d360(x)	Normalize between 0 - 360 and rounds to nearest integer value	<value>!lua return d360(540.7)</value>
dgrd(x)	Convert degrees to radians	<value>!lua return d360(90)</value>
dnor(x)	Normalize between 0 - 360	<value>!lua return dnor(540)</value>
near(x)	Returns x+0.5 if x>0 else returns x-0.5	<value>!lua return near(10.5)</value>
rddg(x)	Radian to degrees	<value>!lua return rddg(pi)</value>
rnor(x)	Normalize between 0 - 2pi	<value>!lua return rnor(4 * pi())</value>
sqrt(x)	sqrt(x)	<value>!lua return sqrt(9)</value>
scmp(x,y)	String comparison (strcmp)	<value>!lua return scmp("equal", "Equal")</value>
scmi(x,y)	String comparison ignoring case (strcmp)	<value>!lua return scmi("equal", "Equal")</value>
varget	Gets value of a sim variable	<value>!lua return varget("P:Local time", "hours")</value>
varset	Sets value of a sim variable	<value>!lua return varset("K:PARKING_BRAKES", "1")</value>
varsswitch	Switch statement. First variable is the control variable, subsequent variables are each case.	<value>!lua return varsswitch(num_engines, 0, 50, 110, 250)</value>
soundOneShot	Plays the specified sound once.	<value>!lua return soundOneShot("canopy.wav")</value>
soundCreate	Creates a sound associated with the specified sound file. Returns an ID which can be used to configure the sound using soundSetProperties.	<value>!lua local var canopySoundID = soundCreate("canopy.wav")</value>
soundSetProperties	Configures a sound to be played or stopped. Also specifies whether the sound should loop. The ID to be used is returned from soundCreate. The first parameter is the ID. The second parameter is true for play and false for stop. The third parameter is true for looping and false for single playback.	<value>!lua soundSetProperties(canopySoundID, true, true)</value>

Examples

This is a simple clock example:

```

<Gauge Name="lua_clock" Version="1.0">
  <Image Name="clock_background.bmp"/>
  <!-- ====== Seconds Hand
  ====== -->
  <Element>
    <Position X="53.8" Y="95.4" />
    <Image Name="clock_second_hand.bmp" PointsTo="North">
      <Axis X="3.8" Y="40.4" />
    </Image>
    <Rotate>
      <Value>!lua
        return flr(varget("P:Local time", "seconds")) %
60
      </Value>
      <Nonlinearity>
        <Item Value="0" Degrees="-90" />
        <Item Value="15" Degrees="0" />
        <Item Value="30" Degrees="90" />
        <Item Value="45" Degrees="180" />
        <Item Value="60" Degrees="270" />
      </Nonlinearity>
    </Rotate>
  </Element>
  <!-- ====== Hour Hand
  ====== -->
  <Element>
    <Position X="53.6" Y="95.4" />
    <Image Name="clock_hour_needle.bmp" PointsTo="West">
      <Axis X="29.6" Y="3.39999" />
    </Image>
    <Rotate>
      <Value>!lua
        return varget("P:Local time", "hours")
      </Value>
    </Rotate>
  </Element>
</Gauge>

```

```

<Nonlinearity>
    <Item Value="0" Degrees="-90" />
    <Item Value="3" Degrees="0" />
    <Item Value="6" Degrees="90" />
    <Item Value="9" Degrees="180" />
    <Item Value="12" Degrees="270" />
</Nonlinearity>
</Rotate>
</Element>
<!-- ===== Minute Hand -->
<Element>
    <Position X="53.8" Y="95.4" />
    <Image Name="clock_minute_needle.bmp" PointsTo="North">
        <Axis X="4.8" Y="41.4" />
    </Image>
    <Rotate>
        <Value>!lua
            return flr(varget("P:Local time", "minutes"))
        </Value>
        <Nonlinearity>
            <Item Value="0" Degrees="-90" />
            <Item Value="15" Degrees="0" />
            <Item Value="30" Degrees="90" />
            <Item Value="45" Degrees="180" />
            <Item Value="60" Degrees="270" />
        </Nonlinearity>
    </Rotate>
</Element>
<Mouse>
    <Help ID="HELPID_GAUGE_CLOCK_WITH_TIME" />
    <Tooltip ID="TOOLTIPTEXT_CLOCK" />
</Mouse>
</Gauge>

```

For a more advanced example of the power of lua script, here is a script used in Prepar3D's example radar gauge which is used for panning and zooming the radar map using the mouse. When the radar map is not zoomed in, this script will set the cursor to the position the user clicks on. When the map is zoomed in, the mouse is used to pan around using a click and drag. At high visual zoom levels, the data zoom level is set to match which sharpens the radar image. Finally the scroll wheel can be used to zoom in and out. For more information on the Prepar3D radar service and how to use it via XML, refer to the [this article](#).

Note that "--" indicates a comment in Lua.

```

<Script>!lua
    event = varget("M:Event","String")
    cursorSet = 0
    -- Handle Cursor set or scroll
    if event == "LeftSingle" or event == "LeftDrag" or event == "LeftRelease" then
        x = varget("M:X","Number") / 255.0
        y = varget("M:Y","Number") / 255.0
        --
        Zoomed in center cursor on view and pan around on drag
        z = varget("C:P3DRadar:VisualZoom","Number")
        if z > 1 then
            if event == "LeftDrag" then
                lx = varget("L:P3DRadarLastCursorPositionX","Number")
                ly = varget("L:P3DRadarLastCursorPositionY","Number")
                cx = varget("C:P3DRadar:CursorPositionX","Number")
                cy = varget("C:P3DRadar:CursorPositionY","Number")
                finalx = cx - ( x - lx ) / z
                finaly = cy - ( y - ly ) / z
                varset("C:P3DRadar:CursorPositionX",finalx)
                varset("C:P3DRadar:CursorPositionY",finaly)
                cursorSet = 1
            elseif event == "LeftRelease" then
                -- set freeze mode back to previous value
                varset("C:P3DRadar:FreezeEnabled",varget("L:P3DRadarFreezeEnabled","Number"));
            else
                --
                freeze image while panning because data texture
                -- Will get remapped if data zoom is used
                varset("C:P3DRadar:FreezeEnabled",1);
            end
            --
            store last cursor position to compare against for dragging
            varset("L:P3DRadarLastCursorPositionX",x)
            varset("L:P3DRadarLastCursorPositionY",y)
        else
            --
            not zoomed so just set cursor position directly
            varset("C:P3DRadar:CursorPositionX",x)
            varset("C:P3DRadar:CursorPositionY",y)
            cursorSet = 1
        end
        --
        if either case above was hit where cursor got set and tracking is enabled
        --
        then request the lat and lon of the cursor because the radar system doesn't
        --
        keep the lat lon around. This will get set in the update script
        if cursorSet == 1 and varget("L:P3DRadarTrackEnabled","Number") == 1 then
            varset("L:P3DRadarTrackLat","Number", varget("C:P3DRadar:CursorPositionLat","Number"))
            varset("L:P3DRadarTrackLon","Number", varget("C:P3DRadar:CursorPositionLon","Number"))

```

```
    end

    elseif event == "WheelUp" then
        -- zoom in 0.25 when mouse wheel scrolls up
        z = varget("C:P3DRadar:VisualZoom", "Number")
        if z < 10 then
            z = z + 0.25
            if z > 3 then
                varset("C:P3DRadar:DataZoom", z)
            else
                varset("C:P3DRadar:DataZoom", 1)
            end
            varset("C:P3DRadar:VisualZoom", z)
        end
    elseif event == "WheelDown" then
        -- zoom out 0.25 when mouse wheel scrolls down
        z = varget("C:P3DRadar:VisualZoom", "Number")
        if z > 1 then
            z = z - 0.25
            if z > 3 then
                varset("C:P3DRadar:DataZoom", z)
            else
                varset("C:P3DRadar:DataZoom", 1)
            end
            varset("C:P3DRadar:VisualZoom", z)
        end
    end

```

</Script>

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

SimConnect

Contents

- [SimConnect API Reference](#)
- [SimConnect Projects](#)
- [SimConnect Samples](#)
- [SimConnect Configuration](#)

Related Links

- [SDK Overview](#)
- [Add-ons Overview](#)

Overview

The SimConnect API can be used by programmers to write [software add-on components](#) for *Prepar3D*. Add-on components for *Prepar3D* can be written in C, C++, or, if the managed API calls are being used, any .NET language such as C#.net or VB.net. Typically the components will perform one or more of the following:

- Add the processing for a new complex gauge, or other instrument, to *Prepar3D*.
 - Replace *Prepar3D* processing of one or more events with new logic.
 - Record or monitor a scenario.
 - Extend the scenario system of *Prepar3D*.
 - Create and set the flight plans for AI (non-user) aircraft.
 - Set different weather systems.
 - Enable new hardware to work with *Prepar3D*.
 - Control an additional camera that the user can optionally select to view.
-

SimConnect API Reference

The SimConnect API is organized by category and can be viewed in the [References Overview](#) article.

SimConnect Projects

Overview

The design of a SimConnect [software add-on components](#) involves writing a client to communicate with a server running within *Prepar3D*. The client opens up communications with the server, then requests which events and object information should be passed to it. The client then waits for the request to be received from the server, and then processes its response appropriately.

The recommended method of writing a [software add-on component](#) is to build it out-of-process, as an application (i.e. an .exe file) rather than in-process, as a library (i.e. a .dll file). This is because out-of-process applications provide more stability, if they crash they will typically will not crash *Prepar3D*, and are easier to build, test and debug. Out-of-process also supports managed code, and therefore applications can be written in .NET languages which ease building complex user interfaces and can expedite the development process.

To get started, learn about how to create either native or managed SimConnect applications:

- [C++ SimConnect Projects](#)
- [Managed SimConnect Projects](#)

Tutorial

For a quickstart, follow alongside these tutorials on creating SimConnect applications:

- [C++ SimConnect Project Tutorial](#)

Design Considerations

SimConnect makes extensive use of ID numbers defined by the client. There are ID numbers for requests, data definitions, events, groups, and so on. These ID numbers should be unique for the client. Re-using an ID will result in the previous call using that ID becoming obsolete, and ignored by the server.

Only one version of *Prepar3D* can be running on one computer at a time, so a client installed on a local machine will only be communicating with one server.

However it is possible to have one client communicate with multiple copies of *Prepar3D*, running on a network.

SimConnect Samples

A list of native and managed samples which showcase SimConnect's functionality can be found in the [Samples Overview](#) article.

SimConnect Configuration

SimConnect supports configuring both the SimConnect Client and the SimConnect Server via respective configuration files. Additionally, configuration support exists to support debugging such as logging and a console window. See the [Configuration Files Overview](#) to learn about the supported configuration options.

[- top -](#)

Managed SimConnect Projects

Overview

There is a managed wrapper for SimConnect that enables .NET language programmers to write SimConnect clients. This section describes how to set up a managed client project, and the key differences to look out for when programming to the wrapper. There are also a number of [Managed Code Samples](#) included with the SDK.

Getting Started With Managed SimConnect

1. Ensure the [Microsoft .NET Framework 4.8 Developer Pack](#) is installed as many of the [Managed Samples](#) target this version of the framework.
2. Create a new project using [Visual Studio 2019](#) using one of the following project types:
 - **C#**: WinForms, WPF, Console
 - **VB.NET**: WinForms, WPF, Console
3. Add a new reference to **LockheedMartin.Prepar3D.SimConnect.dll** in your project. This file can be found in this directory of the SDK:
SDKlib\SimConnect\managed
4. Import the following assemblies in the code files using SimConnect functionality:

C#

```
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;
```

VB.NET

```
Imports LockheedMartin.Prepar3D.SimConnect
Imports System.Runtime.InteropServices
```

5. Set up the initialization and deinitialization of the SimConnect Client.

The native function calls [SimConnect_Open](#) and [SimConnect_Close](#) have been replaced by the **SimConnect** constructor, and **Dispose** method respectively. This means that there is no handle required for the function calls, so for the most part the managed calls use the same parameters as the native calls, except without the SimConnect handle. The code to open and close a SimConnect client is:

C#

```
// Open
// Declare a SimConnect object
SimConnect simconnect = null;
// User-defined win32 event
const int WM_USER_SIMCONNECT = 0x0402;
try
{
    simconnect = new SimConnect("Managed Data Request", this.Handle, WM_USER_SIMCONNECT, null, 0);
}
catch (COMException ex)
{
    // A connection to the SimConnect server could not be established
}
What would have been a failed HRESULT returned in the native API translates to a COMException.

// Close
if (simconnect != null)
{
    simconnect.Dispose();
    simconnect = null;
}
```

VB.NET

Rem **Open**

```

Try
    simconnect = New SimConnect(" VB Managed Data Request", Me.Handle,
WM_USER_SIMCONNECT, Nothing, 0)
Catch ex As Exception
    Rem Failed to connect
End Try
What would have been a failed HRESULT returned in the native API translates to a
COMException.

Rem Close
If simconnect IsNot Nothing Then
    simconnect.Dispose()
    simconnect = Nothing
End If

```

6. Recieve messages from the SimConnect server.

The top level **ReceiveDispatch** switch case statement is not necessary. The client should register a handler for the appropriate *OnRecvXXX* event, and call **ReceiveMessage** when it is notified that messages are waiting in the queue. For Windows applications use a win32 handle (a Control.Handle) to SimConnect to receive notifications when a message arrives. The code for this is as follows:

C#: WinForms

```

protected override void DefWndProc(ref Message m)
{
    if (m.Msg == WM_USER_SIMCONNECT)
    {
        if (simconnect != null)
        {
            simconnect.ReceiveMessage();
        }
    }
    else
    {
        base.DefWndProc(ref m);
    }
}

```

C# WPF

```

IntPtr handle;
HwndSource handleSource;

// Register WPF window
YourWPFWindow()
{
    handle = new WindowInteropHelper(this).Handle; // Get handle of main WPF Window
    handleSource = HwndSource.FromHwnd(handle); // Get source of handle in order to add event
    handlers to it
    handleSource.AddHook(HandleSimConnectEvents);
}

~YourWPFWindow()
{
    if (handleSource != null)
    {
        handleSource.RemoveHook(HandleSimConnectEvents);
    }
}

private IntPtr HandleSimConnectEvents(IntPtr hWind, int message, IntPtr wParam, IntPtr lParam,
ref bool isHandled)
{
    isHandled = false;

    switch (message)
    {
        case WM_USER_SIMCONNECT:
        {
            if (simConnect != null)

```

```

    {
        simConnect.ReceiveMessage();
        isHandled = true;
    }
    break;

    default:
        break;
    }

    return IntPtr.Zero;
}

```

VB.NET

```

Protected Overrides Sub DefWndProc(ByRef m As Message)
    If m.Msg = WM_USER_SIMCONNECT Then
        If simconnect IsNot Nothing Then
            simconnect.ReceiveMessage()
        End If
    Else
        MyBase.DefWndProc(m)
    End If
End Sub

```

Managed SimConnect Programming Differences

Besides the differences in initialization, deinitialization, and receiving dispatched messages, there are other nuances to be aware of. This section enumerates key differences to be aware of as a developer:

- Previously, the managed SimConnect assembly was installed in the GAC (global assembly cache) during the installation of the SDK, so it did not have to be manually placed anywhere for your application to run. However, you did need to reference a copy of this dll in Visual Studio so that it could resolve symbols at compile time. As Visual Studio does not support referencing an assembly stored in the GAC, a copy of the SimConnect.dll was shipped solely for this purpose. It is no longer handled this way in Prepar3D.
- The use of raw memory pointers is not permitted in the managed API. You must define your structs and attribute them properly so that the system marshaller can handle them. See the code in the [Managed Data Request](#) sample for C#, and [VB Managed Data Request](#) sample for VB.NET.
- When the client receives either a **SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE** or a **SIMCONNECT_RECV_SIMOBJECT_DATA** structure, the **dwData[0]** member will contain your struct as **System.Object**. This should be cast to the proper type.
- Partial data return (**SIMCONNECT_DATA_REQUEST_FLAG_TAGGED**) is unsupported. For a description of this flag see the [SimConnect>AddToDataDefinition](#) function.
- Constants in the native simconnect.h are declared as static members of the SimConnect class. For example: **SIMCONNECT_OBJECT_ID_USER** becomes **SimConnect.SIMCONNECT_OBJECT_ID_USER**
- Enums in managed code are scoped. The names of the enum members have been changed, for example, **SIMCONNECT_RECV_ID_QUIT** in native code is **SIMCONNECT_RECV_ID.QUIT** in C#, or **LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_RECV_ID.QUIT** in VB.NET. Note that in VB.NET the full reference is necessary to locate a structure or enumeration.
- Structs are mostly unchanged, with the exception that a char array is represented as a **System.String**. However, they do need to be registered with the managed wrapper with a call to **RegisterDataDefineStruct**. If a string is too long during the data marshaling, it will get truncated safely. Some specific information must be provided with each structure, as shown in bold in the example below (also refer to the [Managed Data Request](#) or [VB Managed Data Request](#) samples):

C#

```

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
struct Struct1
{
    // this is how you declare a fixed size string
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public String title;
    public double latitude;
    public double longitude;
    public double altitude;
}

```

```

};

// define the data structure
// Note that the DATATYPE.STRING256 matches the SizeConst 256 in the MarshalAs statement above
//
simconnect.AddToDataDefinition((uint)DEFINITIONS.Struct1, "title", null, SIMCONNECT_DATATYPE.STRING256, 0,
SimConnect.SIMCONNECT_UNUSED);

simconnect.AddToDataDefinition((uint)DEFINITIONS.Struct1, "Plane Latitude", "degrees", SIMCONNECT_DATATYPE.FLOAT64, 0,
SimConnect.SIMCONNECT_UNUSED);

simconnect.AddToDataDefinition((uint)DEFINITIONS.Struct1, "Plane Longitude", "degrees", SIMCONNECT_DATATYPE.FLOAT64, 0,
SimConnect.SIMCONNECT_UNUSED);

simconnect.AddToDataDefinition((uint)DEFINITIONS.Struct1, "Plane Altitude", "feet", SIMCONNECT_DATATYPE.FLOAT64, 0,
SimConnect.SIMCONNECT_UNUSED);

//
// IMPORTANT: register it with the simconnect managed wrapper marshaller
// if you skip this step, you will only receive a uint in the .dwData field.
//
simconnect.RegisterDataDefineStruct<Struct1>((uint)DEFINITIONS.Struct1);

```

VB.NET

```

<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Pack:=1)> _
Structure Struct1
    Rem This is how you declare a fixed size string
    <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=256)> _
    Public title As String
    Public latitude As Double
    Public longitude As Double
    Public altitude As Double
End Structure

Rem define a data structure, note the last parameter, datumID must be different for each item

simconnect.AddToDataDefinition(StructDefinitions.Struct1, "title", "", 
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.STRING256, 0, 0)

simconnect.AddToDataDefinition(StructDefinitions.Struct1, "Plane Latitude", "degrees",
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.FLOAT64, 0, 1)

simconnect.AddToDataDefinition(StructDefinitions.Struct1, "Plane Longitude", "degrees",
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.FLOAT64, 0, 2)

simconnect.AddToDataDefinition(StructDefinitions.Struct1, "Plane Altitude", "feet",
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.FLOAT64, 0, 3)

Rem IMPORTANT: register it with the simconnect managed wrapper marshaller
Rem if you skip this step, you will only receive an int in the .dwData field.
simconnect.RegisterDataDefineStruct(Of Struct1)(StructDefinitions.Struct1)

```

- Variable length strings are not supported in the managed layer.
- Optional parameters are not supported in the managed layer.
- To send data in an array, copy the array to a polymorphic object array. For example, to apply a waypoint list to an AI aircraft, go through the following steps:

C#

```

// Note that the client code should already have declared a
// data definition, DEFINITIONS.1, and
// calledSimConnect_RequestDataOnSimObjectType to return an AI aircraft ID.

// Step 1: Add a the waypoint list to the data definition

simconnect.AddToDataDefinition(DEFINITIONS.1, "AI WAYPOINT LIST", "number",
SIMCONNECT_DATATYPE.WAYPOINT, 0.0f, SimConnect.SIMCONNECT_UNUSED);

// Step 2: Declare an array of the appropriate size

SIMCONNECT_DATA_WAYPOINT[] waypoints = new SIMCONNECT_DATA_WAYPOINT[2];

// Step 3: Populate the array with all the required data

```

```

waypoints[0].Flags = (uint)SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED;
waypoints[0].ktsSpeed = 100;
waypoints[0].Latitude = 10;
waypoints[0].Longitude = 20;
waypoints[0].Altitude = 1000;

waypoints[1].Flags = (uint)SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED;
waypoints[1].ktsSpeed = 150;
waypoints[1].Latitude = 11;
waypoints[1].Longitude = 21;
waypoints[1].Altitude = 2000;

// Step 4: The managed wrapper marshaling code expects a polymorphic array

Object[] objv = new Object[ waypoints.Length ];
waypoints.CopyTo(objv, 0);
```

// Step 5: Now make the call to apply the waypoint structure to the AI aircraft

```

simconnect.SetDataOnSimObject(DEFINITIONS.1, AIircraftID,
SIMCONNECT_DATA_SET_FLAG.DEFAULT, objv);
```

VB.NET

REM Note that the client code should already have declared a
REM data definition, StructDefinitions.DEFINITIONS.1, and
REM called simconnect.RequestDataOnSimObjectType to return an AI aircraft ID.

REM Step 1: Add a the waypoint list to the data definition

```

simconnect.AddToDataDefinition(StructDefinitions.DEFINITION1, "AI WAYPOINT LIST",
"number", LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.WAYPOINT,
0.0F, LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_UNUSED)
```

REM Step 2: Declare an array of the appropriate size

```

Dim waypoints(2) As
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATA_WAYPOINT
```

REM Step 3: Populate the array with all the required data

```

waypoints(0).Flags =
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED
waypoints(0).ktsSpeed = 100
waypoints(0).Latitude = 10
waypoints(0).Longitude = 20
waypoints(0).Altitude = 1000
```

```

waypoints(1).Flags =
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED
waypoints(1).ktsSpeed = 100
waypoints(0).Latitude = 11
waypoints(0).Longitude = 21
waypoints(0).Altitude = 1000
```

REM Step 4: The managed wrapper marshaling code expects a polymorphic array

```

Dim objv(waypoints.Length) As Object
waypoints.CopyTo(objv, 0)
```

REM Step 5: Now make the call to apply the waypoint structure to the AI aircraft

```

simconnect.SetDataOnSimObject(StructDefinitions.DEFINITION1, AIircraftID,
LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATA_SET_FLAG.DEFAULT, objv)
```

C++ SimConnect Projects

Overview

SimConnect Projects

To build SimConnect add-ons, you must use Visual Studio 2019 or later. To build the project make sure you have completed the following steps.

Potential issues when using SimConnect.lib

C++ projects that import SimConnect.lib will need to also import the following libraries:

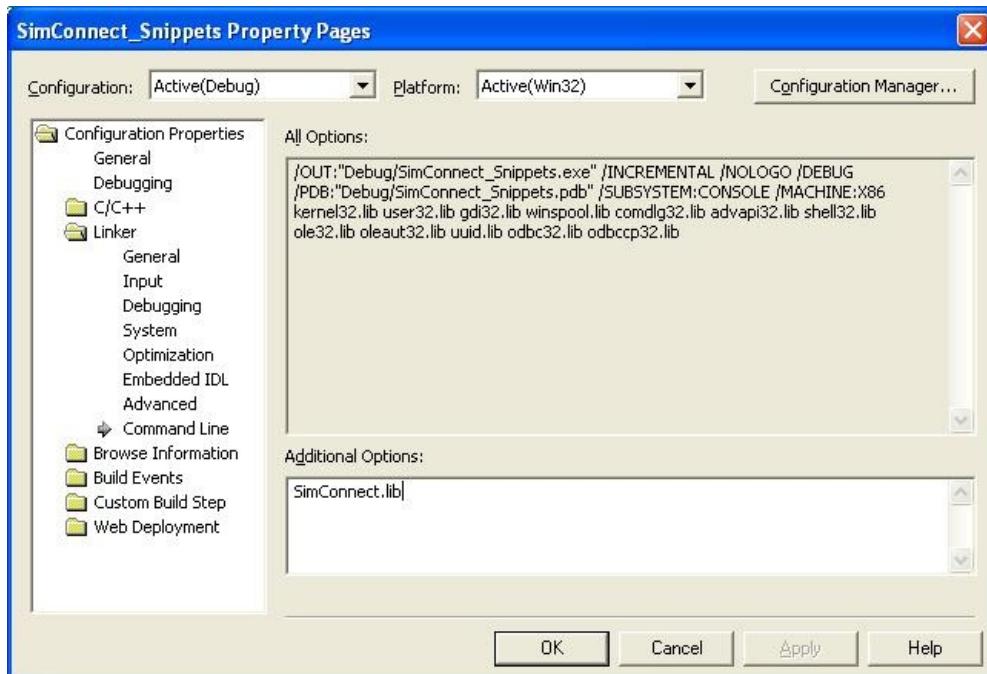
- user32.lib
- AdvApi32.lib
- Shell32.lib

Steps to add:

- Right click on the project
- Expand linker from the left pane
- Click on Input
- Add these libraries in the Additional Dependencies field

C/C++ Projects

1. For C or C++ add-ons, start a new Win32 Console Application project if the add-on will have no user interface. Start a new Win32 Project or MFC Application if the add-on will have a user interface.
2. Ensure the Platform Toolset (Configuration Properties, General) is set to **Visual Studio 2019 (v142)**.
3. Include the **SimConnect.h** header file.
4. Link to the SimConnect.lib library, by adding **SimConnect.lib** to the Additional Options of the Command Line (see image below). If SimConnect.lib is not local to your project, ensure its folder path is included in Additional Library Directories (Configuration Properties, Linker, General).

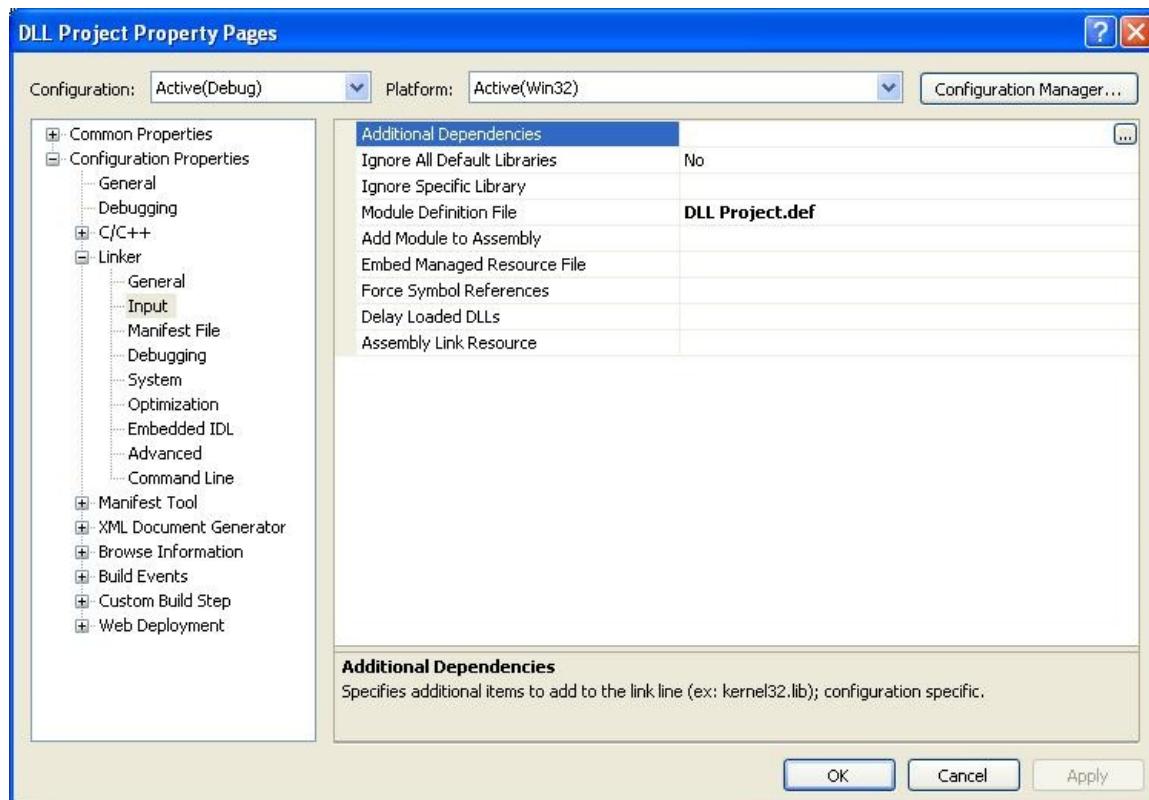


5. Build the application using the function calls described in this document.

DLL Projects

1. Create and add a definition file to the project. This is needed because SimConnect expects the exported names **DLLStart** and **DLLStop** to be undecorated (by default additional characters are added to these names). The definition file can be created by selecting **Project/Add New Item** from the main menu in Visual Studio. Edit the definition (.def) file so it looks like this (where *DLL Project* is the name of your project):

```
LIBRARY "DLL Project"
EXPORTS
DLLStart
DLLStop
```
2. Confirm that the definition file has been added to the project by checking the *Linker\Input* properties:



3. Pay particular attention to the remarks and working samples for [SimConnect_CallDispatch](#).

All Projects

1. Ensure that the Solution Platforms, in Visual Studio 2019 or later, have been set to 64-bit. While SimConnect interfaces are backwards compatible with legacy 32-bit SimConnects, all new development should be done as a x64 solution.
2. Ensure the [SimConnect.ini](#) file is placed in your `%USERPROFILE%\Documents\Prepar3D v5 Files` folder. A default version of this file can be found in the `SDK\config\SimConnect` directory. You do not usually have to make any changes to this file. The `SimConnect.ini` file provided ensures that the SimConnect debug window opens to display communication information between the server and the clients.
3. If the client is to work remotely, write a [SimConnect.cfg](#) file for it.
4. By default, SimConnect is disabled. To enable it, you must copy the necessary configuration files to the correct locations. Default versions of these files can be found in the `SDK\config\SimConnect` directory. Ensure the [EXE.xml](#), [DLL.xml files](#) and [SimConnect.xml](#) files are placed in either the `%APPDATA%\Lockheed Martin\Prepar3D v5` or `%PROGRAMDATA%\Lockheed Martin\Prepar3D v5` folder, and edit these files according to the instructions that follow.
5. Run *Prepar3D* to test your SimConnect client.

Using Legacy SimConnect Versions

Prepar3D

FSX and ESP

Some third party add-on software requires an older version of SimConnect that was built with either FSX or ESP. When installing Prepar3D through the Prepar3D Bundle, the older SimConnect redistributables are installed automatically. To manually install or reinstall the redistributables, go to the `redist\Interface` folder of the installed product. There are multiple different legacy versions of

SimConnect available:

- ESPv1
- FSX-RTM (*Flight Simulator X*)
- FSX-RTM-JPN (*Flight Simulator X, Japanese*)
- FSX-SP1 (*Flight Simulator X: Service Pack 1*)
- FSX-SP1-RUS (*Flight Simulator X: Service Pack 1, Russian*)
- FSX-SP2-XPACK (*Flight Simulator X: Service Pack 2 with Expansion*)

Each of the legacy versions of SimConnect can be installed by following the instructions in their root folders. For most, you can install them using their respective .msi files.

After installing legacy SimConnect client libraries, they will be located in the WinSxS (side-by-side) folder. This means that software add-on components written for older versions of SimConnect will still run as newer versions are released.

[- top -](#)

C++ SimConnect Project Tutorial

Overview

The purpose of this tutorial is to learn how to get started with the [SimConnect Samples](#) that are available in the SDK. Using the [Client Event Sample](#) as the basis of this tutorial, instructions are provided for opening, compiling, running, and understanding the anatomy of a [C++ SimConnect Project](#). Each line of the program's structure will be dissected. Finally, the program will be modified to change its original functionality.

Opening the ClientEvent Project Sample

The [Client Event Sample](#) can be found in the **ClientEvent** folder alongside the rest of the samples in the samples folder:

SDK\SimConnect

The easiest way to open the project is to open the **SDK Samples.sln** solution file:

SDK\SDK Samples.sln

After **Visual Studio 2019** loads the sample projects, select **ClientEvent** in the **Solution Explorer** and use the context menu to select the **Set as StartUp Project** option.

Inspecting the ClientEvent Project Sample's Properties

Before running the sample or modifying it, the project properties will be inspected in order to understand how to correctly include the **SimConnect** library.

Select **Client Event** in the **Solution Explorer** and use the context menu to select the **Properties** option.

- First, set the **Configuration** drop down list to **All Configurations**
- Navigate to **Configuration Properties->General**:
 - Verify that the project's Configuration Type is an **Application (.exe)**
- Navigate to **Configuration Properties->VC++ Directories**:
 - Verify that there is an added **Include Directories** path that contains the directory with the **SimConnect.h** header file:
`..\..\..\Inc\SimConnect\;$(IncludePath)`
 - Verify that there is an added **Library Directories** path that contains the directory with the **SimConnect.lib** library file:
`..\..\..\Lib\SimConnect\;$(LibraryPath)`
- Navigate to **Configuration Properties->Linker->Input**:
 - Verify that **SimConnect.lib** is included along with the automatically inherited libraries:
`SimConnect.lib;<INHERITED LIBRARIES>`
- The SimConnect library will be found because of the additional library path added above.
- Finally, press the **Cancel** button since the purpose of this exercise is merely to inspect the properties and not to change them

Running the ClientEvent Project Sample

NOTE: Before compiling and running the [Client Event Sample](#), make sure that the **Prepar3D** application is running.

To run the [Client Event Sample](#), follow these steps:

- Compile the **ClientEvent** project by right-clicking the **ClientEvent** project in the **Solution Explorer** and selecting **Build** from the context menu
- Run the **ClientEvent** project by pressing the *play icon* which starts the **ClientEvent** application's process and attaches a local windows debugger to it.

This project is programmed to open a **Console Window** when it starts. In Prepar3D, press the key mapped to the vehicle's brakes which for most vehicles this is the **PERIOD** key. After the key is pressed, observe that the **Console Window** displays the current state of the brakes.

Understanding the ClientEvent Project Sample

Connecting to SimConnect

To start communications with the SimConnect server, the [SimConnect_Open](#) function is called. If the function returns a success, denoted by **S_OK**, then communication has been established with the server. If the function is not a success, then the SimConnect server could not be found. For example, Prepar3D may not be running or there was a problem with SimConnect configuration files.

Once the new client has been created and communication has been established, a handle **hSimConnect** is populated and can be used in other SimConnect functions.

```
if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Client Event", NULL, 0, NULL, 0)))
{
    printf("\nConnected to Prepar3D!");
    ...
}
else
    printf("\nFailed to Connect");
```

Listening for a Specified Simulation Event

To map one of the simulation events in Prepar3D to an event defined by the client, use the [SimConnect_MapClientEventToSimEvent](#) function. Event names can be found in the [Event IDs](#) article.

hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES, "brakes");
where **EVENT_BRAKES** is defined as a unique enumeration value:

```
static enum EVENT_ID {
    EVENT_BRAKES,
};
```

For Prepar3D to notify the client that the simulation event has been triggered, the event must be registered to a notification group. To register an event to a notification group and assign it a priority, the following functions must be called: [SimConnect_AddClientEventToNotificationGroup](#) and [SimConnect_SetNotificationGroupPriority](#). Using these functions, the client event can be added to a notification group and that notification group can be given a priority.

```
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_BRAKES);
hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);
where GROUP0 is defined as a unique enumeration value:
static enum GROUP_ID {
    GROUP0,
};
```

Processing SimConnect Messages

To process incoming SimConnect messages from the SimConnect Server, the program enters a processing loop where it continuously calls [SimConnect_CallDispatch](#) to receive messages until the SimConnect Server sends a quit message.

```

while( 0 == quit )
{
    SimConnect_CallDispatch(hSimConnect, MyDispatchProc1, NULL);
    Sleep(1);
}
SimConnect_CallDispatch is supplied a function pointer to MyDispatchProc1 which handles the
receiving of SimConnect Server messages. In MyDispatchProc1, If the client's event called
EVENT_BRAKES is received, then a message is printed to the Console Window. If a quit message
is received, then the message processing loop is stopped.
void CALLBACK MyDispatchProc1(SIMCONNECT_RECV* pData, DWORD cbData, void
*pContext)
{
    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;
            switch(evt->uEventID)
            {
                case EVENT_BRAKES:
                    printf("\nEvent brakes: %d", evt->dwData);
                    break;

                default:
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_QUIT:
        {
            quit = 1;
            break;
        }

        default:
            break;
    }
}

```

After the quit message is received, the client terminates its connection with the SimConnect Server using [SimConnect_Close](#).

hr = SimConnect_Close(hSimConnect);

Modifying the ClientEvent Project Sample

Since the inner workings of **ClientEvent**'s is now known, the final step of this tutorial is to demonstrate mastery by modifying the behavior of the program. The modification will consist of changing the client event being used from *brakes* to the *pause toggle*.

Make the following changes:

- Rename all instances of the **EVENT_BRAKES** enumeration ID to **EVENT_PAUSE_TOGGLE**
- Change the string identifier passed into [SimConnect_MapClientEventToSimEvent](#) from "brakes" to "pause_toggle"
- Change the printf string from "\nEvent brakes: %d" to the following: "\nEvent pause toggle: %d"
- Verify that the **Prepar3D** application is running
- After these changes, recompile the **ClientEvent** project by selecting the **ClientEvent** project in the **Solution Explorer** and pressing the **Build** option in the context menu
- Just as before, verify that the the ClientEvent is the startup project (**Set as StartUp Project** in context menu) and then run the program.
- Observe that the **Console Window** opens
- In Prepar3D, press the key mapped to the pause toggle button which is by default the **P** key. After

the key is pressed, observe that the **Console Window** displays the current pause state.

[- top -](#)

References

Overview

This section lists all the API functions, structures, enumerations, and other coding information necessary to build SimConnect clients.

Information on each API call includes some example code. This code does not form a complete program in any sense, but simply highlights the uses of the API call.

Functions

General Functions

Lifecycle Specific Function Name	Description
SimConnect_Open	Used to send a request to the <i>Prepar3D</i> server to open up communications with a new client.
SimConnect_Close	Used to request that the communication with the server is ended.
DispatchProc*	*Written by the developer of the SimConnect client, as a callback function to handle all the communications with the server.
SimConnect_CallDispatch	Used to process the next SimConnect message received through the specified callback function.
SimConnect_GetNextDispatch	Used to process the next SimConnect message received, without the use of a callback function.
Client Event Specific Function Name	Description
SimConnect_AddClientEventToNotificationGroup	Used to add an individual client defined event to a notification group.
SimConnect_MapClientEventToSimEvent	Used to associate a client defined event ID with an <i>Prepar3D</i> event name.
SimConnect_MapInputEventToClientEvent	Used to connect input events (such as keystrokes, joystick or mouse movements) with the sending of appropriate event notifications.
SimConnect_RemoveClientEvent	Used to remove a client defined event from a notification group.
SimConnect_TransmitClientEvent	Used to request that the <i>Prepar3D</i> server transmit to all SimConnect clients the specified client event. This version of the function provides a 32-bit context data field.
SimConnect_TransmitClientEvent64	Used to request that the <i>Prepar3D</i> server transmit to all SimConnect clients the specified client event. This version of the function provides a 64-bit context data field.
Client Data Specific Function Name	Description
SimConnect_AddToClientDataDefinition	Used to add an offset and a size in bytes, or a type, to a client data definition.
SimConnect_CreateClientData	Used to request the creation of a reserved data area for this client.
SimConnect_MapClientDataNameToID	Used to associate an ID with a named client data area.
SimConnect_RequestClientData	Used to request that the data in an area created by another client be sent to this client.
SimConnect_SetClientData	Used to write one or more units of data to a client data area.
SimConnect_ClearClientDataDefinition	Used to clear the definition of the specified client data.
Notification Group Specific Function Name	Description
SimConnect_AddClientEventToNotificationGroup	Used to add an individual client defined event to a notification group.
SimConnect_ClearNotificationGroup	Used to remove all the client defined events from a notification group.
SimConnect_RequestNotificationGroup	Used to request events are transmitted from a notification group, when the simulation is in Dialog Mode.
SimConnect_SetNotificationGroupPriority	Used to set the priority of a notification group.
Input Event Specific Function Name	Description
SimConnect_MapInputEventToClientEvent	Used to connect input events (such as keystrokes, joystick or mouse movements) with the sending of appropriate event notifications.
SimConnect_RemoveInputEvent	Used to remove an input event from a specified input group object.
SimConnect_RequestJoystickDeviceInfo	Used to request the name and number of currently connected joystick devices.
SimConnect_RequestReservedKey	Used to request a specific keyboard TAB-key combination applies only to this client.

Input Group Specific Function Name	Description
SimConnect_ClearInputGroup	Used to remove all the input events from a specified input group object.
SimConnect_SetInputGroupPriority	Used to set the priority for a specified input group object.
SimConnect_SetInputGroupState	Used to turn requests for input event information from the server on and off.
System Event Function Name	Description
SimConnect_SetSystemEventState	Used to turn requests for event information from the server on and off.
SimConnect_SubscribeToSystemEvent	Used to request that a specific system event is notified to the client.
SimConnect_SubscribeToSystemEventEx	Used to request that a specific system event is notified to the client. Flags allow you to request a blocking callback.
SimConnect_SubscribeToSystemEventW	Used to request that a specific system event is notified to the client. Flags allow you to request a blocking callback. This version of the function supports unicode string results.
SimConnect_UnsubscribeFromSystemEvent	Used to request that notifications are no longer received for the specified system event.
System Event Function Name	Description
SimConnect_SetSystemState	Used to access a number of <i>Prepar3D</i> system components.
SimConnect_RequestSystemState	Used to request information from a number of <i>Prepar3D</i> system components.
SimConnect_RequestSystemStateW	Used to request information from a number of <i>Prepar3D</i> system components.
Data Definition Specific Function Name	Description
SimConnect_AddToDataDefinition	Used to add a <i>Prepar3D</i> simulation variable name to a client defined object definition.
SimConnect_ClearDataDefinition	Used to remove all simulation variables from a client defined object.
Misc Function Name	Description
SimConnect_GetLastSentPacketID	Returns the ID of the last packet sent to the SimConnect server.
SimConnect_RequestSessionDuration	Used to request the duration of the current scenario in seconds.
SimConnect_RequestVersion	Used to request license type and version numbers of P3D and SimConnect.
SimConnect_RequestResponseTimes	Used to provide some data on the performance of the client-server connection
String Specific Function Name	Description
SimConnect_InsertString	Used to assist in adding variable length narrow strings to a structure.
SimConnect_InsertStringW	Used to assist in adding variable length wide strings to a structure.
SimConnect_RetrieveString	Used to assist in retrieving variable length narrow strings from a structure.
SimConnect_RetrieveStringW	Used to assist in retrieving variable length wide strings from a structure.
Synchronous SimConnect Specific Function Name - Overview	Description
SimConnect_RequestSynchronousBlock	Used to request a blocking callback.
SimConnect_SynchronousUnblock	Used to release a blocking callback.
SimConnect_SetSynchronousTimeout	Used to set the error timeout length for blocking callbacks.
SimConnect_SubscribeToSystemEventEx	Used to request that a specific system event is notified to the client. Flags allow you to request a blocking callback.
Menu Specific Function Name	Description
SimConnect_MenuAddItem	Used to add a menu item, associated with a client event. This version of the function provides a 32-bit context data field.
SimConnect_MenuAddItem64	Used to add a menu item, associated with a client event. This version of the function provides a 64-bit context data field.
SimConnect_MenuAddSubItem	Used to add a sub-menu item, associated with a client event. This version of the function provides a 32-bit context data field.
SimConnect_MenuAddSubItem64	Used to add a sub-menu item, associated with a client event. This version of the function provides a 64-bit context data field.
SimConnect_MenuDeleteItem	Used to remove a client defined menu item.
SimConnect_MenuDeleteSubItem	Used to remove a specified sub-menu item.
SimConnect_Text	Used to display a text menu, message window, or scrolling or static text, on the screen.

SimObject Functions

General SimObject Function Name	Description
SimConnect_SetDataOnSimObject	Used to make changes to the data properties of an object.
SimConnect_ChangeVehicle	Used to change the user's vehicle.
SimConnect_RequestAttachPointData	Used to request attach point positions and orientations from a simulation object.
SimConnect_RequestDataOnSimObject	Used to request when the SimConnect client is to receive data values for a specific object.
SimConnect_RequestDataOnSimObjectType	Used to retrieve information about simulation objects of a given type that are within a specified radius of the user's vehicle.
AI Object Specific Function Name	Description
SimConnect_AICreateEnrouteATCAircraft	Used to create an AI controlled aircraft that is about to start or is already underway on its flight plan.
SimConnect_AICreateEnrouteATCAircraftW	Used to create an AI controlled aircraft that is about to start or is already underway on its flight plan. This version of the functions supports unicode file paths, titles, tail numbers, and flight plans.
SimConnect_AICreateNonATCAircraft	Used to create an aircraft that is not flying under ATC control (so is typically flying under VFR rules).
SimConnect_AICreateParkedATCAircraft	Used to create an AI controlled aircraft that is currently parked and does not have a flight plan.
SimConnect_AICreateSimulatedObject	Used to create AI controlled objects other than aircraft.
SimConnect_AICreateSimulatedObjectEx	Used to create AI controlled objects other than aircraft. Extended for use with disabled simulations and Owned objects
SimConnect_AICreateObjectWithExternalSim	Used to create an AI object, overriding the sim to use a specific External Sim instead.
SimConnect_AIReleaseControl	Used to clear the AI control of a simulated object, typically an aircraft, in order for it to be controlled by a SimConnect client.
SimConnect_AIReleaseControlEx	Used to clear the AI control of a simulated object, typically an aircraft, in order for it to be controlled by a SimConnect client. Extended to be able to remove the AI entirely.
SimConnect_AIRemoveObject	Used to remove any object created by the client using one of the SimConnect AI creation functions.
SimConnect_AISetAircraftFlightPlan	Used to set or change the flight plan of an AI controlled aircraft.
SimConnect_AISetAircraftFlightPlanW	Used to set or change the flight plan of an AI controlled aircraft. This version of the function supports unicode file paths.
SimConnect_AISetGroundClamp	Used to enable/disable ground clamping on a specific SimConnect owned SimObject.
Attach Sim Specific Function Name - Overview	Description
SimConnect_AttachObjectToSimObject	Used to create a SimObject and attach it to an existing SimObject.
SimConnect_AttachSimObjectToSimObject	Used to attach an existing SimObject to another existing SimObject.
SimConnect_ReleaseSimObjectFromSimObject	Used to release a SimObject from another SimObject.
Attach Weapon Specific Function Name - Overview	Description
SimConnect_AttachWeaponToObject	Used to create a Weapon and attach it to an existing SimObject.
SimConnect_ClearWeapons	Used to remove all Weapons from a SimObject.
External Sim Specific Function Name - Overview	Description
SimConnect_RegisterExternalSim	Used to register this client as an external sim provider.
SimConnect_UnregisterExternalSim	Used to unregister this client as an external sim provider.
SimConnect_RegisterExternalSecondarySim	Used to register this client as an

SimConnect_RegisterExternalSecondarySim	external secondary sim provider.
SimConnect_UnregisterExternalSecondarySim	Used to unregister this client as an external secondarysim provider.
SimConnect_AttachExternalSecondarySimToSimObject	Used to attach an External Secondary Sim to an existing SimObject.
SimConnect_DetachExternalSecondarySimFromSimObject	Used to detach an External Secondary Sim from an existing SimObject.
SimConnect_AICreateObjectWithExternalSim	Used to create an AI object, overriding the sim to use a specific External Sim instead.
SimConnect_ChangeVehicleWithExternalSim	Used to change the user's vehicle and override the sim to use a specific External Sim instead.

View Functions

View Specific Function Name	Description
SimConnect_ChangeView	Used to change the camera used by the main view.
SimConnect_OpenView	Used to open a new view.
SimConnect_CloseView	Used to close a view.
SimConnect_DockView	Used to dock a view.
SimConnect_UndockView	Used to undock a view.
Camera Post-Process Specific Function Name	Description
SimConnect_AddPostProcess	Used to add a post process effect to the specified camera.
SimConnect_AddPostProcessMainCamera	Used to add a post process effect to the main camera.
SimConnect_RemovePostProcess	Used to remove a post process effect to the specified camera.
SimConnect_RemovePostProcessMainCamera	Used to remove a post process effect to the main camera.
Camera Sensor Specific Function Name	Description
SimConnect_RequestCameraSensorMode	Used to request the current sensor mode of a given view.
SimConnect_RequestMainCameraSensorMode	Used to request the current sensor mode of the main view.
SimConnect_SetCameraSensorMode	Used to change the sensor mode of a camera.
SimConnect_SetMainCameraSensorMode	Used to change the sensor mode of the main camera.
Camera Specific Function Name	Description
SimConnect_CameraPanToView	Used to smoothly transition the view of the specified camera to the view of the specified target camera. This function currently only works for Custom Cameras .
SimConnect_CameraSetRelative6DOF	Used to adjust the user's aircraft view camera.
SimConnect_CameraSetRelative6DofByName	Used to adjust the specified view's camera.
SimConnect_CameraSmoothRelative6DOF	Used to smoothly transition the view of the main camera to the specified location.
SimConnect_CameraSmoothRelative6DOFByName	Used to smoothly transition the view of the specified camera to the specified location.
SimConnect_CameraZoomIn	Used to adjust the zoom of the camera.
SimConnect_CameraZoomOut	Used to adjust the zoom of the camera.
SimConnect_CreateCameraDefinition	Used to create a new camera definition.
SimConnect_CreateCameraInstance	Used to create a new camera instance from a camera definition.
SimConnect_DeleteCameraInstance	Used to remove a camera instance from a camera definition.
SimConnect_MainCameraPanToView	Used to smoothly transition the view of the main camera to the view of the specified target camera. This function currently only works for Custom Cameras .
SimConnect_MainCameraZoomIn	Used to adjust the zoom of the main camera.
SimConnect_MainCameraZoomOut	Used to adjust the zoom of the main camera.
SimConnect_RequestCameraFov	Used to request the field-of-view of a given view (horizontal and vertical). See SIMCONNECT_RECV_CAMERA_FOV .
SimConnect_RequestCameraRelative6DOF	Used to request the XYZ delta offset from the eyepoint reference point, as well as the pitch, bank, and heading of the main view.
SimConnect_RequestCameraRelative6DofByName	Used to request the XYZ delta offset from the eyepoint reference point, as well as the pitch, bank, and heading of a given view.
	I used to request the current window position

SimConnect_RequestCameraWindowPosition	Used to request the current window position of a given view.
SimConnect_RequestCameraWindowSize	Used to request the current window size of a given view.
SimConnect_RequestMainCameraFov	Used to request the field-of-view of the main view (horizontal and vertical). See SIMCONNECT_RECV_CAMERA_FOV .
SimConnect_SendCameraCommand	Used to simulate user inputs to control camera movement and rotation.
SimConnect_SetCameraFov	Used to change the FoV (Field of View) of a camera.
SimConnect_SetCameraHorizontalFov	Used to change the horizontal FoV (Field of View) of a camera.
SimConnect_SetCameraVerticalFov	Used to change the vertical FoV (Field of View) of a camera.
SimConnect_SetCameraWindowPosition	Used to set the window position of the specified view.
SimConnect_SetCameraWindowSize	Used to set the window size of the specified view.
SimConnect_SetMainCameraFov	Used to change the FoV (Field of View) of the main camera.
SimConnect_SetMainCameraHorizontalFov	Used to change the horizontal FoV (Field of View) of the main camera.
SimConnect_SetMainCameraVerticalFov	Used to change the vertical FoV (Field of View) of the main camera.
Image and Video Capture Specific Function Name	
SimConnect_CaptureImage	Used to capture an image of a view.
SimConnect_CaptureImageW	Used to capture an image of a view. This version of the function supports unicode file paths, filenames, and view names.
SimConnect_BeginVideoStream	Used to begin a video stream of a view.
SimConnect_EndVideoStream	Used to end a video stream of a view.
Observer View Specific Function Name	
SimConnect_CreateObserver	Used to create an observer view.
SimConnect_RequestObserverData	Used to request all data for a specific observer.
SimConnect_MoveObserver	Used to move an observer using an xyz translation.
SimConnect_RotateObserver	Used to rotate an observer around a specified axis.
SimConnect_SetObserverPosition	Used to place an observer at a specific world position.
SimConnect_SetObserverRotation	Used to set an observer's specific rotation.
SimConnect_SetObserverLookAt	Used to have an observer look at a world position.
SimConnect_SetObserverLookAtEx	Used to have an observer look at a world position round world corrected.
SimConnect_ObserverTrackLocationOn	Used to have an observer track a world position (round world corrected) using an above ground level altitude and accounting for changes in elevation.
SimConnect_ObserverTrackLocationOff	Used to disable continuous tracking of the previously specified world position.
SimConnect_SetObserverFieldOfView	Used to set an observer's field of view.
SimConnect_SetObserverStepSize	Used to set an observer's linear and angular step size.
SimConnect_SetObserverFocalLength	Used to set an observer's focal length (in meters).
SimConnect_SetObserverFocusFixed	Used to switch an observer's focus between local and world.
SimConnect_SetObserverRegime	Used to switch an observer's regime (terrestrial, tellurian, or ghost).
SimConnect_SetObserverZoomLevels	Used to set an observer's zoom level.
SimConnect_ObserverTrackEntityOn	Used to enable continuous tracking of the specified entity.
SimConnect_ObserverTrackEntityOff	Used to disable continuous tracking of the previously specified entity.
SimConnect_ObserverAttachToEntityOn	Used to attach an observer to the specified entity.
SimConnect_ObserverAttachToEntityOff	Used to detach an observer from its attached entity.
SimConnect_SetObserverSceneryOrigin	Used to switch an observer's scenery origin mode (target or self).
Deprecated Camera Function Name	
SetCameraRenderSettings	Deprecated
SetCameraColorizationMode	Deprecated

World Functions

Setting Specific Function Name	Description
--------------------------------	-------------

SimConnect_RequestShadowFlags	Used to request the current shadow flag settings.
SimConnect_RequestTrafficSettings	Used to request the current traffic settings.
SimConnect_SetTrafficSettings	Used to set the current traffic settings.
SimConnect_RequestSceneryComplexity	Used to request the current scenery complexity setting.
Facility Data Function Name	Description
SimConnect_RequestFacilitiesList	Used to request a list of all the facilities of a given type currently held in the facilities cache.
SimConnect_SubscribeToFacilities	Used to request notifications when a facility of a certain type is added to the facilities cache.
SimConnect_UnsubscribeToFacilities	Used to request that notifications of additions to the facilities cache are not longer sent.
Ground Info Specific Function Name - Overview	Description
SimConnect_RequestGroundInfo	Used to request a grid of ground altitudes at a specific lat/lon.
SimConnect_RequestGroundInfoOnSimObject	Used to request a recurring grid of ground altitudes based on a SimObjects lat/lon.
Weather Specific Function Name	Description
SimConnect_WeatherCreateStation	Used to add a weather station.
SimConnect_WeatherCreateThermal	Used to create a thermal at a specific location.
SimConnect_WeatherRemoveStation	Used to remove a weather station.
SimConnect_WeatherRemoveThermal	Used to remove a thermal.
SimConnect_WeatherRequestCloudState	Used to request cloud density information on a given area.
SimConnect_WeatherRequestInterpolatedObservation	Used to send a request for weather data that is interpolated from the weather at the nearest weather stations.
SimConnect_WeatherRequestObservationAtNearestStation	Used to send a request for the weather data from the weather station nearest to the specified lat/lon position.
SimConnect_WeatherRequestObservationAtStation	Used to send a request for the weather data from a weather station identified by its ICAO code.
SimConnect_WeatherSetDynamicUpdateRate	Used to set the rate at which cloud formations change.
SimConnect_WeatherSetModeCustom	Used to set the weather mode to user-defined.
SimConnect_WeatherSetModeGlobal	Used to set the weather mode to global, so the same weather data is used everywhere.
SimConnect_WeatherSetModeServer	Deprecated.
SimConnect_WeatherSetModeTheme	Used to set the weather mode to a particular theme.
SimConnect_WeatherSetObservation	Used to set the weather at a specific weather station, identified from within the Metar data string
Effect Specific Function Name	Description
SimConnect_CreateEffect	Used to create an effect at a location or attach it to an existing SimObject.
SimConnect_RemoveEffect	Used to remove an effect.
Scenario Functions	
Scenario Functions	Description
SimConnect_FlightLoad	Used to load an existing scenario file.
SimConnect_FlightLoadW	Used to load an existing scenario file. This version of the function supports unicode file paths.
SimConnect_FlightPlanLoad	Used to load an existing flight plan.
SimConnect_FlightPlanLoadW	Used to load an existing flight plan. This version of the function supports unicode file paths.
SimConnect_FlightSave	Used to save the current state of a scenario to a scenario file.
SimConnect_FlightSaveW	Used to save the current state of a scenario to a scenario file. This version of the function supports unicode file paths, titles, and descriptions.

Structured Scenario Specific Function Name	Description
SimConnect_CompleteCustomMissionAction	Used to complete the scenario action specified by a GUID.
SimConnect_ExecuteMissionAction	Used to execute the scenario action specified by a GUID.
SimConnect_RequestFlightSegmentCount	Used to request the number of Flight Segment objects in the active scenario.
SimConnect_RequestFlightSegmentDataByGUID	Used to request information about a Flight Segment object in the active scenario.
SimConnect_RequestFlightSegmentDataByIndex	Used to request information about a Flight Segment object in the active scenario.
SimConnect_RequestFlightSegmentRangeData	Used to request information about a specific range of a Flight Segment object in the active scenario.
SimConnect_RequestGoalCount	Used to request the number of Goal/Group Goal objects in the active scenario.
SimConnect_RequestGoalDataByGUID	Used to request information about a Goal/Group Goal object in the active scenario.
SimConnect_RequestGoalDataByIndex	Used to request information about a Goal/Group Goal object in the active scenario.
SimConnect_RequestMissionObjectiveCount	Used to request the number of Mission Objective objects in the active scenario.
SimConnect_RequestMissionObjectiveDataByGUID	Used to request information about a Goal/Group Goal object in the active scenario.
SimConnect_RequestMissionObjectiveDataByIndex	Used to request information about a Goal/Group Goal object in the active scenario.
SimConnect_ResolveGoal	Used to resolve a goal to specified goal state.
SimConnect_RequestChildGoalDataByIndex	Used to request information about a mission objective or group goal's children.
SimConnect_RequestLandingTriggerCount	Used to request the number of Area Landing Trigger or Airport Landing Trigger objects in the active scenario.
SimConnect_RequestLandingTriggerLandingInfoCount	Used to request information about a Landing Trigger object's landings in the active scenario.
SimConnect_RequestLandingTriggerLandingInfoByIndex	Used to request information about a specific landing of a Landing Trigger object in the active scenario.
SimConnect_RequestMobileSceneryInRadius	Used to retrieve mobile scenery objects that are within a specified radius of the user's vehicle.
SimConnect_RequestMobileSceneryDataByID	Used to retrieve data about a specific mobile scenery object using its object ID.

Analysis Functions

Flight Analysis Specific Function Name	Description
SimConnect_GenerateFlightAnalysisDiagrams	Used to generate diagrams as shown in the flight analysis UI which can be used, for example, to examine ILS landing performance.
Recorder Specific Function Name	Description
SimConnect_PlaybackRecording	Used to playback a specified recording from a specified start time to a specified end time.
SimConnect_PlaybackRecordingW	Used to playback a specified recording from a specified start time to a specified end time. This version of the function supports unicode file paths.
SimConnect_StartRecorder	Used to trigger the Prepar3D Recorder to start recording.
SimConnect_StopRecorderAndSaveRecording	Used to trigger the Prepar3D Recorder to stop recording and either prompt the user or save the recording with a specified file name.
SimConnect_RequestRecordingInfo	Used to request information on a recording.
SimConnect_RequestRecordingInfoW	Used to request information on a recording. This version of the function supports unicode file paths.
SimConnect_RequestBookmarkInfo	Used to request bookmark information on a recording.
SimConnect_RequestBookmarkInfoW	Used to request bookmark information on a recording. This version of the function supports

SimConnect Structures and Enumerations

SimConnect uses the following structures and enumerations.

Name	Structure or Enumeration	Description
SIMCONNECT_CLIENT_DATA_PERIOD	Enum	Used with the SimConnect_RequestClientData call to specify how often data is to be sent to the client.
SIMCONNECT_DATA_RACE_RESULT	Struct	Used to hold multiplayer racing results.
SIMCONNECT_DATATYPE	Enum	Used with the SimConnect_AddToDataDefinition call to specify the data type that the server should use to return the specified data to the client.
SIMCONNECT_DATA_GROUND_INFO	Struct	Used to return info about one ground point entry in the array returned by a call to SimConnect_RequestGroundInfo or SimConnect_RequestGroundInfoOnSimObject
SIMCONNECT_DATA_FACILITY_AIRPORT	Struct	Used to return information on a single airport in the facilities cache.
SIMCONNECT_DATA_FACILITY_NDB	Struct	Used to return information on a single NDB station in the facilities cache.
SIMCONNECT_DATA_FACILITY_VOR	Struct	Used to return information on a single VOR station in the facilities cache.
SIMCONNECT_DATA_FACILITY_TACAN	Struct	Used to return information on a single TACAN station in the facilities cache.
SIMCONNECT_DATA_FACILITY_WAYPOINT	Struct	Used to return information on a single waypoint in the facilities cache.
SIMCONNECT_DATA_INITPOSITION	Struct	Used to initialize the position of a the user or AI controlled aircraft.
SIMCONNECT_DATA_JOYSTICK_DEVICE_INFO	Struct	Used to return information about connected joystick devices by a call to SimConnect_RequestJoystickDeviceInfo
SIMCONNECT_DATA_LATLONALT	Struct	Used to hold a world position.
SIMCONNECT_DATA_MARKERSTATE	Struct	Used to help graphically link flight model data with the graphics model.
SIMCONNECT_DATA_WAYPOINT	Struct	Used to hold all the necessary information on a waypoint.
SIMCONNECT_DATA_XYZ	Struct	Used to hold a 3D co-ordinate.
SIMCONNECT_DATA_PBH	Struct	Used to hold a rotation.
SIMCONNECT_DATA_OBSERVER	Struct	Used to hold all data for a specific observer.
SIMCONNECT_DATA_OBJECT_DAMAGED_BY_WEAPON	Struct	Used to hold all data for an object damaged by a weapon.
SIMCONNECT_DATA_VIDEO_STREAM_INFO	Struct	Used to hold all data for a video stream.
SIMCONNECT_EXCEPTION	Enum	Used with the SIMCONNECT_RECV_EXCEPTION structure to return information on an error that has occurred.
SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG	Enum	Used with the SimConnect_RegisterExternalSim and SimConnect_RegisterExternalSecondarySim functions to specify which External Sim Callbacks you wish to receive.
SIMCONNECT_GOAL_RESOLUTION	Enum	Used by SimConnect_ResolveGoal to specify the state the specified goal should be resolved to.
SIMCONNECT_GOAL_STATE	Enum	Used by SIMCONNECT_RECV_GOAL to specify the state of the goal.
SIMCONNECT_GROUND_INFO_LATLON_FORMAT	Enum	Used with the Simconnect_RequestGroundInfo and SimConnect_RequestGroundInfoOnSimObject functions to specify the format for latitude and longitude values.
SIMCONNECT_GROUND_INFO_ALT_FORMAT	Enum	Used with the Simconnect_RequestGroundInfo and SimConnect_RequestGroundInfoOnSimObject functions to specify the format for altitude values.
SIMCONNECT_GROUND_INFO_SOURCE_FLAG	Enum	Used with the Simconnect_RequestGroundInfo and SimConnect_RequestGroundInfoOnSimObject functions to specify types of ground you wish to receive.
SIMCONNECT_FACILITY_LIST_TYPE	Enum	Used to determine which type of facilities data is being requested or returned.
SIMCONNECT_LICENSE_TYPE	Enum	Used to determine which type of License is being run on the Application.
SIMCONNECT_MISSION_END	Enum	Used to specify the three possible outcomes of a scenario.
SIMCONNECT_MISSION_OBJECT_TYPE	Enum	Returned by SIMCONNECT_RECV_MISSION_OBJECT_COUNT struct to specify the type of scenario object.
SIMCONNECT_MISSION_OBJECTIVE_STATUS	Enum	Used by SIMCONNECT_RECV_MISSION_OBJECTIVE to specify the state of the mission objective.
SIMCONNECT_PERIOD	Enum	Used with the SimConnect_RequestDataOnSimObject call to specify how often data is to be sent to the client.
SIMCONNECT_RECV	Struct	Used with the SIMCONNECT_RECV_ID enumeration to indicate which type of structure has been returned.
SIMCONNECT_RECV_AIRPORT_LIST	Struct	Used to return a list of SIMCONNECT_DATA_FACILITY_AIRPORT structures.
SIMCONNECT_RECV_ASSIGNED_OBJECT_ID	Struct	Used to return an object ID that matches a request ID.
SIMCONNECT_RECV_ATTACHPOINT_DATA	Struct	Will be received by the client after a successful call to SimConnect_RequestAttachPointData .
SIMCONNECT_RECV_CAMERA_6DOF	Struct	Used to return the XYZ delta offset from the eyepoint reference point in meters as well as the pitch, bank, and heading of the camera.
SIMCONNECT_RECV_CAMERA_FOV	Struct	Used to return the camera's current horizontal and vertical field of

SIMCONNECT_RECV_CAMERA_FOV	Struct	view..
SIMCONNECT_RECV_CAMERA_SENSOR_MODE	Struct	Used to return the camera's current sensor mode..
SIMCONNECT_RECV_CAMERA_WINDOW_POSITION	Struct	Used to return the camera's current window position.
SIMCONNECT_RECV_CAMERA_WINDOW_SIZE	Struct	Used to return the camera's current window size.
SIMCONNECT_RECV_CLIENT_DATA	Struct	Will be received by the client after a successful call to SimConnect_RequestClientData . The structure is identical to SIMCONNECT_RECV_SIMOBJECT_DATA .
SIMCONNECT_RECV_CLOUD_STATE	Struct	Used to return an array of cloud state data.
SIMCONNECT_RECV_CUSTOM_ACTION	Struct	Used specifically with the scenario system, providing details on the custom action that has been triggered.
SIMCONNECT_RECV_EVENT	Struct	Used to return an event ID to the client.
SIMCONNECT_RECV_EVENT_64	Struct	Used to return an event ID along with 64-bit user context data to the client.
SIMCONNECT_RECV_EVENT_FILENAME	Struct	Used with the SimConnect_SubscribeToSystemEvent to return a filename and an event ID to the client.
SIMCONNECT_RECV_EVENT_FRAME	Struct	Used with the SimConnect_SubscribeToSystemEvent to return the frame rate and simulation speed to the client.
SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED	Struct	Sent to a client when they have successfully joined a multiplayer race.
SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED	Struct	Sent to the host when the session is visible to other users in the lobby.
SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED	Struct	Sent to a client when they have requested to leave a race, or to all players when the session is terminated by the host.
SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE	Struct	Used to return the type and ID of an AI object that has been added or removed from the simulation, by any client.
SIMCONNECT_RECV_EVENT_RACE_END	Struct	Used to hold the results for one player at the end of a race.
SIMCONNECT_RECV_EVENT_RACE_LAP	Struct	Used to hold the results for one player at the end of a lap.
SIMCONNECT_RECV_EXCEPTION	Struct	Used with the SIMCONNECT_EXCEPTION enumeration type to return information on an error that has occurred.
SIMCONNECT_RECV_EXTERNAL_SIM_BASE	Struct	Base class for External Sim callback data.
SIMCONNECT_RECV_EXTERNAL_SIM_CREATE	Struct	Used with the External Sim Create Callback.
SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY	Struct	Used with the External Sim Destroy Callback.
SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE	Struct	Used with the External Sim Simulate Callback.
SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED	Struct	Used with the External Sim Location Changed Callback.
SIMCONNECT_RECV_EXTERNAL_SIM_EVENT	Struct	Used with the External Sim Event Callback.
SIMCONNECT_RECV_EVENT_WEAPON	Struct	Used to send a notification of a detachable weapon being fired or detonated.
SIMCONNECT_RECV_EVENT_COUNTERMEASURE	Struct	Used to send a notification of a deployed countermeasure.
SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON	Struct	Used to send a notification of an object being damaged by a weapon.
SIMCONNECT_RECV_FACILITIES_LIST	Struct	Used to provide information on the number of elements in a list of facilities returned to the client, and the number of packets that were used to transmit the data.
SIMCONNECT_RECV_FLIGHT_SEGMENT	Struct	Received by the client after a successful call to SimConnect_RequestFlightSegmentDataByGUID or SimConnect_RequestFlightSegmentDataByIndex .
SIMCONNECT_RECV_FLIGHT_SEGMENT_READY_FOR_GRADING	Struct	Received by the client with a FlightSegmentReadyForGrading notification. Contains instance ID of the Flight Segment.
SIMCONNECT_RECV_GOAL	Struct	Received by the client after a successful call to SimConnect_RequestGoalDataByGUID or SimConnect_RequestGoalDataByIndex .
SIMCONNECT_RECV_GOAL_PAIR	Struct	Received by the client after a successful call to SimConnect_RequestChildGoalDataByIndex .
SIMCONNECT_RECV_MISSION_OBJECT_COUNT	Struct	Received by the client after a successful call to SimConnect_RequestFlightSegmentCount , SimConnect_RequestGoalCount , SimConnect_RequestMissionObjectiveCount , or SimConnect_RequestLandingTriggerCount .
SIMCONNECT_RECV_MISSION_OBJECTIVE	Struct	Received by the client after a successful call to SimConnect_RequestMissionObjectiveDataByGUID or SimConnect_RequestMissionObjectiveDataByIndex .
SIMCONNECT_RECV_PARAMETER_RANGE	Struct	Received by the client after a successful call to SimConnect_RequestFlightSegmentRangeData .
SIMCONNECT_RECV_PLAYBACK_STATE_CHANGED	Struct	Received by the client with a PlaybackStateChanged notification. Contains whether playback has started or ended and the file name of the recording.
SIMCONNECT_RECV_RECORDER_STATE_CHANGED	Struct	Received by the client with a RecorderStateChanged notification. Contains whether recording has started or ended..
SIMCONNECT_RECV_RECORDING_INFO	Struct	Received by the client after a successful call to SimConnect_RequestRecordingInfo
SIMCONNECT_RECV_RECORDING_BOOKMARK_INFO	Struct	Received by the client after a successful call to SimConnect_RequestBookmarkInfo
SIMCONNECT_RECV_LANDING_TRIGGER_INFO	Struct	Received by the client after a successful call to SimConnect_RequestLandingTriggerLandingInfoCount
SIMCONNECT_RECV_LANDING_INFO	Struct	Received by the client after a successful call to SimConnect_RequestLandingTriggerLandingInfoByIndex
SIMCONNECT_RECV_GROUND_INFO	Struct	Used with the SimConnect_RequestGroundInfo and SimConnect_RequestGroundInfoOnSimObject functions.
SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO	Struct	Received by the client after a successful call to SimConnect_RequestJoystickDeviceInfo

SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS	Struct	Received by the client after a successful call to SimConnect_RequestMobileSceneryInRadius
SIMCONNECT_RECV_MOBILE_SCENERY_DATA	Struct	Received by the client after a successful call to SimConnect_RequestMobileSceneryDataByID
SIMCONNECT_RECV_ID	Enum	Used within the SIMCONNECT_RECV structure to indicate which type of structure has been returned.
SIMCONNECT_RECV_NDB_LIST	Struct	Used to return a list of SIMCONNECT_DATA_FACILITY_NDB structures.
SIMCONNECT_RECV_OPEN	Struct	Used to return information to the client, after a successful call to SimConnect_Open .
SIMCONNECT_RECV_QUIT	Struct	This is an identical structure to the SIMCONNECT_RECV structure.
SIMCONNECT_RECV_RESERVED_KEY	Struct	Used with the SimConnect_RequestReservedKey function to return the reserved key combination.
SIMCONNECT_RECV_SCENERY_COMPLEXITY	Struct	Will be received by the client after a successful call to SimConnect_RequestSceneryComplexity .
SIMCONNECT_RECV_SESSION_DURATION	Struct	Will be received by the client after a successful call to SimConnect_RequestSessionDuration .
SIMCONNECT_RECV_SHADOW_FLAGS	Struct	Will be received by the client after a successful call to SimConnect_RequestShadowFlags .
SIMCONNECT_RECV_SIMOBJECT_DATA	Struct	Will be received by the client after a successful call to SimConnect_RequestDataOnSimObject or SimConnect_RequestDataOnSimObjectType .
SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE	Struct	Will be received by the client after a successful call to SimConnect_RequestDataOnSimObjectType . The structure is identical to SIMCONNECT_RECV_SIMOBJECT_DATA .
SIMCONNECT_RECV_SYNCHRONOUS_BLOCK	Struct	Will be received by the client in response to a call to SimConnect_RequestSynchronousBlock .
SIMCONNECT_RECV_SYSTEM_STATE	Struct	Used with the SimConnect_RequestSystemState function to retrieve specific Prepar3D systems states and information.
SIMCONNECT_RECV_TACAN_LIST	Struct	Used to return a list of SIMCONNECT_DATA_FACILITY_TACAN structures.
SIMCONNECT_RECV_TRAFFIC_SETTINGS	Struct	Will be received by the client after a successful call to SimConnect_RequestTrafficSettings .
SIMCONNECT_RECV_VOR_LIST	Struct	Used to return a list of SIMCONNECT_DATA_FACILITY_VOR structures.
SIMCONNECT_RECV_WAYPOINT_LIST	Struct	Used to return a list of SIMCONNECT_DATA_FACILITY_WAYPOINT structures.
SIMCONNECT_RECV_WEATHER_OBSERVATION	Struct	Used to return weather observation data after a call to SimConnect_WeatherRequestInterpolatedObservation , SimConnect_WeatherRequestObservationAtStation , or SimConnect_WeatherRequestObservationAtNearestStation .
SIMCONNECT_RECV_OBSERVER_DATA	Struct	Will be received by the client after a successful call to SimConnect_RequestObserverData .
SIMCONNECT_SIMOBJECT_TYPE	Enum	Used with the SimConnect_RequestDataOnSimObjectType call to request information on specific or nearby objects.
SIMCONNECT_STATE	Enum	Used with the SimConnect_SetSystemEventState call to turn the reporting of events on and off.
SIMCONNECT_TEXT_RESULT	Enum	Used to specify which event has occurred as a result of a call to SimConnect_Text .
SIMCONNECT_TEXT_TYPE	Enum	Used to specify which type of text is to be displayed by the SimConnect_Text function
SIMCONNECT_WAYPOINT_FLAGS	Enum	Used with the SIMCONNECT_DATA_WAYPOINT structure to define waypoints.
SIMCONNECT_WEATHER_MODE	Enum	Used to return the current weather mode, after a call using the SIMCONNECT_RECV_ID_EVENT_WEATHER_MODE setting.
SIMCONNECT_CAMERA_SENSOR_MODE	Enum	Used to specify a camera's sensor mode using the SimConnect_SetCameraSensorMode call.
SIMCONNECT_CAMERA_TYPE	Enum	Used with the SimConnect_CreateCameraDefinition call to create a new camera definition.
SIMCONNECT_CAMERA_COMMAND	Enum	Used with the SimConnect_SendCameraCommand call to move camera.
SIMCONNECT_DYNAMIC_FREQUENCY	Enum	Used with the SimConnect_SetTrafficSettings function and the SIMCONNECT_RECV_TRAFFIC_SETTINGS struct.
SIMCONNECT_SCENERY_COMPLEXITY	Enum	Used with the SimConnect_RequestSceneryComplexity function and the SIMCONNECT_RECV_SCENERY_COMPLEXITY struct.

- top -

General Functions

Overview

To view a list of all general SimConnect functions, see the [General Functions](#) table.

DispatchProc

The **DispatchProc** function is a written by the developer of the SimConnect client, as a callback function to handle all the communications with the server.

Syntax

```
void CALLBACK DispatchProc(  
    SIMCONNECT_RECV* pData,  
    DWORD cbData,  
    void * pContext  
)
```

Parameters

pData

[in] Pointer to a data buffer, to be treated initially as a [SIMCONNECT_RECV](#) structure. If you are going to make a copy of the data buffer (which is maintained by the SimConnect client library) make sure that the defined buffer is large enough (the size of the returned data structure is one member of the [SIMCONNECT_RECV](#) structure).

cbData

[in] The size of the data buffer, in bytes.

pContext

[in] Contains the pointer specified by the client in the [SimConnect_CallDispatch](#) function call.

Return Values

This function does not return a value.

Example

```
void CALLBACK MyDispatchProc(SIMCONNECT_RECV* pData, DWORD cbData)  
{  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_OPEN:  
            // enter code to handle SimConnect version information received in a  
            // SIMCONNECT\_RECV\_OPEN structure.  
            SIMCONNECT_RECV_OPEN *openData = (SIMCONNECT_RECV_OPEN*) pData;  
            break;  
  
        case SIMCONNECT_RECV_ID_EVENT:  
            // enter code to handle events received in a SIMCONNECT\_RECV\_EVENT structure.  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*) pData;  
            break;  
  
        case SIMCONNECT_RECV_ID_EVENT_64:  
            // enter code to handle events received in a SIMCONNECT\_RECV\_EVENT\_64 structure.  
            SIMCONNECT_RECV_EVENT_64 *evt = (SIMCONNECT_RECV_EVENT_64*) pData;  
            break;  
  
        case SIMCONNECT_RECV_ID_EVENT_FILENAME:  
            // enter code to handle event filenames received in a  
            // SIMCONNECT\_RECV\_EVENT\_FILENAME  
            // structure.  
            SIMCONNECT_RECV_EVENT_FILENAME *evt =  
                (SIMCONNECT_RECV_EVENT_FILENAME*) pData;  
            break;  
    }  
}
```

```

case SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE:
    // enter code to handle AI objects that have been added or removed, and received in a
SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE
    // structure.
    SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE *evt =
(SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE*) pData;
    break;

case SIMCONNECT_RECV_ID_EVENT_FRAME:
    // enter code to handle frame data received in a SIMCONNECT_RECV_EVENT_FRAME
    // structure.
    SIMCONNECT_RECV_EVENT_FRAME *evt = (SIMCONNECT_RECV_EVENT_FRAME*)
pData;
    break;

case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:
    // enter code to handle object data received in a SIMCONNECT_RECV_SIMOBJECT_DATA
structure.
    SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData =
(SIMCONNECT_RECV_SIMOBJECT_DATA*) pData;
    break;

case SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE:
    // enter code to handle object data received in a
SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE
    // structure.
    SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE *pObjData =
(SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE*) pData;
    break;

case SIMCONNECT_RECV_ID_QUIT:
    // enter code to handle exiting the application
    break;

case SIMCONNECT_RECV_ID_EXCEPTION:
    // enter code to handle errors received in a SIMCONNECT_RECV_EXCEPTION structure.
    SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)
pData;
    break;

case SIMCONNECT_RECV_ID_WEATHER_OBSERVATION:
    // enter code to handle object data received in a
SIMCONNECT_RECV_WEATHER_OBSERVATION structure.
    SIMCONNECT_RECV_WEATHER_OBSERVATION* pWxData =
(SIMCONNECT_RECV_WEATHER_OBSERVATION*) pData;
    const char* pszMETAR = (const char*)(pWxData+1);
    break;

// Enter similar case statements to handle the other types of data that can be received, including:
// SIMCONNECT_RECV_ID_ASSIGNED_OBJECT_ID,
// SIMCONNECT_RECV_ID_RESERVED_KEY,
// SIMCONNECT_RECV_ID_CUSTOM_ACTION
// SIMCONNECT_RECV_ID_SYSTEM_STATE
// SIMCONNECT_RECV_ID_CLOUD_STATE

default:
    // enter code to handle the case where an unexpected message is received
    break;
}
}

```

Working Samples

Primary samples [Client Event](#)
[Tracking Errors](#)

Reference samples All but a few of the other [samples](#) implement this function.

Remarks

This function can be named appropriately by the client developer. The name of the function is passed to the client-side library with the [SimConnect_CallDispatch](#) function call. Handle all the callback

events in this function. If you do not wish to implement a callback function use [SimConnect_GetNextDispatch](#).

To receive time based notifications, see the [SimConnect_SubscribeToSystemEvent](#) function. To receive event based notifications see the [SimConnect_AddClientEventToNotificationGroup](#) function. To send an event to be received by other clients, see the [SimConnect_TransmitClientEvent](#) function.

See Also

- [SimConnect_CallDispatch](#)
 - [SimConnect API Reference](#)
-

SimConnect_AddClientEventToNotificationGroup

The **SimConnect_AddClientEventToNotificationGroup** function is used to add an individual client defined event to a notification group.

Syntax

```
HRESULT SimConnect_AddClientEventToNotificationGroup(  
    HANDLE hSimConnect,  
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID,  
    SIMCONNECT_CLIENT_EVENT_ID EventID,  
    BOOL bMaskable = FALSE  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

GroupID

[in] Specifies the ID of the client defined group.

EventID

[in] Specifies the ID of the client defined event.

bMaskable

[in, optional] Boolean, **True** indicates that the event will be masked by this client and will not be transmitted to any more clients, possibly including *Prepar3D* itself (if the priority of the client exceeds that of *Prepar3D*). **False** is the default. See the explanation of [SimConnect Priorities](#).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum EVENT_ID {  
    EVENT_1,  
    EVENT_2  
    EVENT_3  
};  
static enum GROUP_ID {  
    GROUP_1,  
};  
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_1);  
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_2);  
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_3,  
    TRUE);  
hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_1,  
    SIMCONNECT_GROUP_PRIORITY_HIGHEST);
```

Working Samples

	Client Event
	Cockpit Camera
	Joystick Input
Primary samples	Menu Items
	Send Event A
	Send Event B
	Send Event C
	Tracking Errors

Reference samples Many of the other [samples](#) implement this function.

Remarks

The maximum number of events that can be added to a notification group is 1000. A notification group is simply a convenient way of setting the appropriate priority for a range of events, and all client events (such as EVENT_1, EVENT_2, EVENT_3 in the example above) must be assigned to a notification group before any event notifications will be received from the SimConnect server.

See Also

- [SimConnect API Reference](#)
- [SimConnect_RemoveClientEvent](#)
- [SimConnect_SetNotificationGroupPriority](#)
- [SimConnect_ClearNotificationGroup](#)

SimConnect_AddToClientDataDefinition

The **SimConnect_AddToClientDataDefinition** function is used to add an offset and a size in bytes, or a type, to a client data definition.

Syntax

```
HRESULT SimConnect_AddToClientDataDefinition(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_DATA_DEFINITION_ID DefineID,
    DWORD dwOffset,
    DWORD dwSizeOrType,
    float fEpsilon = 0,
    DWORD DatumID = SIMCONNECT_UNUSED
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

DefineID

[in] Specifies the ID of the client-defined client data definition.

dwOffset

[in] Double word containing the offset into the client area, where the new addition is to start. Set this to **SIMCONNECT_CLIENTDATAOFFSET_AUTO** for the offsets to be calculated by the SimConnect server.

dwSizeOrType

[in] Double word containing either the size of the client data in bytes, or one of the following defined values (note that these definitions have a negative value, all positive values will be treated as a size parameter).

Constant	Value
SIMCONNECT_CLIENTDATATYPE_INT8	-1
SIMCONNECT_CLIENTDATATYPE_INT16	-2
SIMCONNECT_CLIENTDATATYPE_INT32	-3
SIMCONNECT_CLIENTDATATYPE_INT64	-4
SIMCONNECT_CLIENTDATATYPE_FLOAT32	-5
SIMCONNECT_CLIENTDATATYPE_FLOAT64	-6

fEpsilon

[in, optional] If data is requested only when it changes (see the *flags* parameter of [SimConnect_RequestClientData](#)), a change will only be reported if it is greater than the value of this parameter (not greater than or equal to). The default is zero, so even the tiniest change will

initiate the transmission of data. Set this value appropriately so insignificant changes are not transmitted. This can be used with integer data, the floating point *fEpsilon* value is first truncated to its integer component before the comparison is made (for example, an *fEpsilon* value of 2.9 truncates to 2, so a data change of 2 will not trigger a transmission, and a change of 3 will do so). This parameter only applies if one of the six constant values listed above has been set in the *dwSizeOrType* parameter, if a size has been specified SimConnect has no record of the type of data being sent, so cannot do a meaningful comparison of values.

DatumID

[in, optional] Specifies a client defined datum ID. The default is zero. Use this to identify the data received if the data is being returned in tagged format (see the *flags* parameter of [SimConnect_RequestClientData](#)). There is no need to specify datum IDs if the data is not being returned in tagged format.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

This function must be called before a client data area can be written to or read from. Typically this function would be called once for each variable that is going to be read or written. Note that an error will not be given if the size of a data definition exceeds the size of the client area -- this is to allow for the case where definitions are specified by one client before the relevant client area is created by another.

The most efficient use of client data areas is to group data that changes at the same time into the same data area. Minor performance improvements are gained by not using tagged data, or the *fEpsilon* parameter, if they are not needed.

See Also

- [SimConnect_ClearClientDataDefinition](#)
- [SimConnect_CreateClientData](#)
- [SimConnect_MapClientDataNameToID](#)
- [SimConnect_RequestClientData](#)
- [SimConnect_SetClientData](#)
- [SimConnect API Reference](#)

SimConnect_AddToDataDefinition

The **SimConnect_AddToDataDefinition** function is used to add a *Prepar3D* simulation variable name to a client defined object definition.

Syntax

```
HRESULT SimConnect_AddToDataDefinition(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_DEFINITION_ID DefineID,
    const char* DatumName,
    const char* UnitsName,
    SIMCONNECT_DATATYPE DatumType = SIMCONNECT_DATATYPE_FLOAT64,
    float fEpsilon = 0,
    DWORD DatumID = SIMCONNECT_UNUSED
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
DefineID
[in] Specifies the ID of the client defined data definition.
DatumName

[in] Specifies the name of the *Prepar3D* simulation variable. See the [Simulation Variables](#) document for a table of variable names. If an index is required then it should be appended to the variable name following a colon, see the example for DEFINITION_2 below. Indexes are numbered from 1 (not zero). Simulation variable names are not case-sensitive (so can be entered in upper or lower case).

UnitsName

[in] Specifies the units of the variable. See the [Simulation Variables](#) document for a table of acceptable unit names. It is possible to specify different units to receive the data in, from those specified in the Simulation Variables document. See DEFINITION_2 below for an example. The alternative units must come under the same heading (such as Angular Velocity, or Volume, as specified in the Units of Measurement section of the Simulation Variables document). For strings and structures enter "NULL" for this parameter.

DatumType

[in, optional] One member of the [**SIMCONNECT_DATATYPE**](#) enumeration type. This parameter is used to determine what datatype should be used to return the data. The default is **SIMCONNECT_DATATYPE_FLOAT64**. Note that the structure data types are legitimate parameters here.

fEpsilon

[in, optional] If data is requested only when it changes (see the *flags* parameter of [SimConnect_RequestDataOnSimObject](#)), a change will only be reported if it is greater than the value of this parameter (not greater than or equal to). The default is zero, so even the tiniest change will initiate the transmission of data. Set this value appropriately so insignificant changes are not transmitted. This can be used with integer data, the floating point *fEpsilon* value is first truncated to its integer component before the comparison is made (for example, an *fEpsilon* value of 2.9 truncates to 2, so a data change of 2 will not trigger a transmission, and a change of 3 will do so).

DatumID

[in, optional] Specifies a client defined datum ID. The default is zero. Use this to identify the data received if the data is being returned in tagged format (see the *flags* parameter of [SimConnect_RequestDataOnSimObject](#)). There is no need to specify datum IDs if the data is not being returned in tagged format.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum DATA_DEFINE_ID {
    DEFINITION_1,
    DEFINITION_2
};

static enum DATA_REQUEST_ID {
    REQUEST_1,
    REQUEST_2,
};

struct Struct1
{
    double kohlsmann;
    double altitude;
    double latitude;
    double longitude;
};

// Match string definitions from the Simulation Variables document with the client defined ID

hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Kohlsman setting
hg", "inHg");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Indicated Altitude",
"feet");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Latitude",
"degrees");
```

```

hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Longitude",
"degrees");

hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_2, "GENERAL ENG
RPM:1", "rpm");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_2, "GENERAL ENG
RPM:2", "revolutions per minute");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_2, "GENERAL ENG
RPM:3", "degrees per second");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_2, "GENERAL ENG
RPM:4", "minutes per round");

// Sections of code in DispatchProc

// At the right point request the data
// In this example the data is being requested on the user aircraft
.....
hr = SimConnect_RequestDataOnSimObject(hSimConnect, REQUEST_1,
DEFINITION_1,SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD_ONCE);

.....
// When the data is received -- cast it to the correct structure type

case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:
{
    SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData =
(SIMCONNECT_RECV_SIMOBJECT_DATA*)pData;

    switch(pObjData->dwRequestID)
    {
        case REQUEST_1:
            Struct1 *pS = (Struct1*)&pObjData->dwData;

            // Add code to process the structure appropriately

            break;
    }
    break;
}
.....

```

Working Samples

Primary samples	Request Data Set Data Tagged Data Throttle Control
------------------------	---

Remarks

The maximum number of entries in a data definition is 1000.

See Also

- [SimConnect API Reference](#)
- [SimConnect_ClearDataDefinition](#)
- [SimConnect_RequestDataOnSimObject](#)
- [SimConnect_RequestDataOnSimObjectType](#)

SimConnect_CallDispatch

The **SimConnect_CallDispatch** function is used to process the next SimConnect message received, through the specified callback function.

Syntax

HRESULT SimConnect_CallDispatch(

```
HANDLE hSimConnect,
DispatchProc pfcnDispatch,
void * pContext
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
pfcnDispatch
[in] Specifies the callback function. For a definition of the function see [DispatchProc](#).
pContext
[in] Specifies a pointer that the client can define that will be returned in the callback. This is used in particular by managed code clients to pass a *this* pointer to the callback.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
int quit = 0;
...
//
while( quit == 0 )
{
    hr = SimConnect_CallDispatch(hSimConnect, MyDispatchProc, NULL);
};
```

Working Samples

Primary samples [Client Event](#)
[Tracking Errors](#)

Reference samples All but a few of the other [samples](#) implement this function.

Remarks

It is important to call this function sufficiently frequently that the queue of information received from the server is processed (typically it is coded within a **while** loop that terminates when the application is exited). However, if the project involves developing a library (DLL) rather than an application (EXE) then only one call to this function is necessary. This call will store the name of the callback in a cache, and whenever a packet is sent to the client, the callback function will be run. The format of a DLL project is shown in the following table:

```
// Include files are no different than for an EXE

HANDLE hSimConnect = NULL;

void CALLBACK MyDispatchProcDLL(SIMCONNECT_RECV* pData, DWORD cbData, void
*pContext)
{
    // Callback code for a DLL is no different than for an EXE
}

//
// The DLLStart function must be present.
//
int __stdcall DLLStart(void)
{
    HRESULT hr;
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "DLL name", NULL, 0, NULL, 0)))
}
```

```

{
    printf("\nConnected...");

    // Place all initialization code for the client in this function

    hr = SimConnect_CallDispatch(hSimConnect, MyDispatchProcDLL, NULL);
}
return 0;
}

// 
// The DLLStop function must be present.
//
void __stdcall DLLStop(void)
{
    // Close the client
    if (hSimConnect != NULL)
        HRESULT hr = SimConnect_Close(hSimConnect);
}

```

In order to implement SimConnect as a class, and maintain an object-oriented design of an add-on, the following constructs can be used. Note in particular the static function and the use of the *pContext* parameter.

```

Class CName
{
    void Dispatch();
    static void DispatchCallback(SIMCONNECT_RECV *pData, DWORD cbData, void *pContext);
    void Process(SIMCONNECT_RECV *pData, DWORD cbData);

    HANDLE hSimConnect; // use SimConnect_Open to set this value.
};

void CName::Dispatch()
{
    ::SimConnect_Dispatch(hSimConnect, &CName::DispatchCallback, this);
}

// static function
void CName::DispatchCallback(SIMCONNECT_RECV *pData, DWORD cbData, void *pContext)
{
    CName *pThis = reinterpret_cast<CName*>(pContext);
    pThis->Process(pData, cbData);
}

void CName::Process(SIMCONNECT_RECV *pData, DWORD cbData)
{
    // do processing of SimConnect data
}

```

See Also

- [SimConnect API Reference](#)
 - [SimConnect_GetNextDispatch](#)
-

SimConnect_ClearClientDataDefinition

The **SimConnect_ClearClientDataDefinition** function is used to clear the definition of the specified client data.

Syntax

```

HRESULT SimConnect_ClearClientDataDefinition(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_DATA_DEFINITION_ID DefineID
);

```

Parameters

hSimConnect

[in] Handle to a SimConnect object.
DefineID
[in] Specifies the ID of the client defined client data definition.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_AddToClientDataDefinition](#)
- [SimConnect_CreateClientData](#)
- [SimConnect_MapClientDataNameToID](#)
- [SimConnect_RequestClientData](#)
- [SimConnect_SetClientData](#)
- [SimConnect API Reference](#)

SimConnect_ClearDataDefinition

The **SimConnect_ClearDataDefinition** function is used to remove all simulation variables from a client defined data definition.

Syntax

```
HRESULT SimConnect_ClearDataDefinition(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_DEFINITION_ID DefineID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
DefineID
[in] Specifies the ID of the client defined data definition.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum DATA_DEFINE_ID {  
    DEFINITION_1,  
    DEFINITION_2  
};  
....  
  
// Match string definitions from the Simulation Variables document with the client defined ID  
  
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Kohlsman setting hg",  
    "inHg");  
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Indicated Altitude", "feet");
```

```
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Latitude", "degrees");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Longitude", "degrees");
...
hr = SimConnect_ClearDataDefinition(hSimConnect, DEFINITION_1);
...
```

Remarks

Use this function to permanently delete a data definition. To temporarily suspend data requests see the remarks for the [SimConnect_RequestDataOnSimObject](#) function.

See Also

- [SimConnect API Reference](#)
- [SimConnect_AddToDataDefinition](#)
- [SimConnect_RequestDataOnSimObject](#)
- [SimConnect_RequestDataOnSimObjectType](#)

SimConnect_ClearInputGroup

The **SimConnect_ClearInputGroup** function is used to remove all the input events from a specified input group object.

Syntax

```
HRESULT SimConnect_ClearInputGroup(
    HANDLE hSimConnect
    SIMCONNECT_INPUT_GROUP_ID GroupID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
GroupID
[in] Specifies the ID of the client defined input group that is to have all its events removed.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum INPUT_ID {
    INPUT_1,
};
static enum EVENT_ID {
    EVENT_1,
};
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1, "parking_brakes");
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "a+B", EVENT_1);
...
hr = SimConnect_ClearInputGroup(hSimConnect, INPUT_1);
```

Remarks

Use this function to permanently delete an input group. Use the [SimConnect_SetInputGroupState](#) function to temporarily suspend input group notifications.

See Also

- [SimConnect API Reference](#)

-
- [SimConnect_MapInputEventToClientEvent](#)
 - [SimConnect_SetInputGroupPriority](#)
 - [SimConnect_RemoveInputEvent](#)
 - [SimConnect_SetInputGroupState](#)
-

SimConnect_ClearNotificationGroup

The **SimConnect_ClearNotificationGroup** function is used to remove all the client defined events from a notification group.

Syntax

```
HRESULT SimConnect_ClearNotificationGroup(  
    HANDLE hSimConnect,  
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID  
)
```

Parameters

hSimConnect
 [in] Handle to a SimConnect object.
GroupID
 [in] Specifies the ID of the client defined group that is to have all its events removed.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum EVENT_ID {  
    EVENT_1,  
    EVENT_2  
    EVENT_3  
};  
static enum GROUP_ID {  
    GROUP_1,  
};  
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_1);  
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_2);  
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_3, TRUE);  
  
hr = SimConnect_ClearNotificationGroup(hSimConnect, GROUP_1);
```

Remarks

There is a maximum of 20 notification groups in any SimConnect client. Use this function if the maximum has been reached, but one or more are no longer required.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AddClientEventToNotificationGroup](#)
 - [SimConnect_RemoveClientEvent](#)
 - [SimConnect_SetNotificationGroupPriority](#)
-

SimConnect_Close

The **SimConnect_Close** function is used to request that the communication with the server is ended.

Syntax

```
HRESULT SimConnect_Close(
    HANDLE hSimConnect
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
hr = SimConnect_Close(hSimConnect);
```

Working Samples

Primary samples [Open and Close](#)
[Windows Event](#)

Reference samples All of the other [samples](#) implement this function.

Remarks

When a SimConnect client is closed, the server will remove all objects, menu items, group definitions and so on, defined or requested by that client, so there is no need to remove them explicitly in the client code.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_Open](#)
-

SimConnect_CreateClientData

The **SimConnect_CreateClientData** function is used to request the creation of a reserved data area for this client.

Syntax

```
HRESULT SimConnect_CreateClientData(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_DATA_ID ClientDataID,
    DWORD dwSize,
    SIMCONNECT_CREATE_CLIENT_DATA_FLAG Flags
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ClientDataID

[in] ID of the client data area. Before calling this function, call

[SimConnect_MapClientDataNameToID](#) to map an ID to a unique client area name.

dwSize

[in] Double word containing the size of the data area in bytes.

Flags

[in] Specify the flag **SIMCONNECT_CREATE_CLIENT_DATA_FLAG_READ_ONLY** if the data area can only be written to by this client (the client creating the data area). By default other

clients can write to this data area.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Use this function, along with the other client data functions, to reserve an area of memory for client data on the server, that other clients can have read (or read and write) access to. Specify the contents of the data area with the [SimConnect_AddToClientDataDefinition](#) call, and set the actual values with a call to [SimConnect_SetClientData](#). Other clients can receive the data with a call to [SimConnect_RequestClientData](#).

The maximum size of a client data area is set by the constant **SIMCONNECT_CLIENTDATA_MAX_SIZE**. For the RTM version of *Prepar3D* this is 8K. There is no maximum number of client data areas, but the total must not exceed 1Mbyte for the RTM version. If a request is made for a client data area greater than **SIMCONNECT_CLIENTDATA_MAX_SIZE** a **SIMCONNECT_EXCEPTION_INVALID_DATA_SIZE** exception is returned. If a request is made for a client data area that will exceed the total maximum memory a **SIMCONNECT_EXCEPTION_OUT_OF_BOUNDS** exception is returned.

One client area can be referenced by any number of client data definitions. Typically the name of the client area, and the data definitions, should be published appropriately so other clients can be written to use them. Care should be taken to give the area a unique name.

Once created, a client data area cannot be deleted or reduced in size. To increase the size of the data area, first close the connection, then re-open it and request the client data area again, using the same name, but with the required size. The additional data area will be initialized to zero, but the previous data will be untouched by this process. Client data persists to the end of the *Prepar3D* session, and is not removed when the client that created it is closed. It is also possible to change a read-only data area to writeable using this technique.

See Also

- [SimConnect_AddToClientDataDefinition](#)
 - [SimConnect_ClearClientDataDefinition](#)
 - [SimConnect_MapClientDataNameToID](#)
 - [SimConnect_RequestClientData](#)
 - [SimConnect_SetClientData](#)
 - [SimConnect API Reference](#)
-

SimConnect_GetLastSentPacketID

The **SimConnect_GetLastSentPacketID** function returns the ID of the last packet sent to the SimConnect server.

Syntax

```
HRESULT SimConnect_GetLastSentPacketID(
    HANDLE hSimConnect,
    DWORD* pdwSendID
);
```

Parameters

hSimConnect
[In] Handle to a SimConnect object.
pdwSendID
[out] Pointer to a double word containing the ID of the last sent packet.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
DWORD dwLastID;

hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MY_EVENT,
"Custom.Event");
hr = SimConnect_TransmitClientEvent(hSimConnect, 0, EVENT_MY_EVENT, 0,
SIMCONNECT_GROUP_PRIORITY_HIGHEST, 0);
// Get the Send ID of the last transmission to the server
hr = SimConnect_GetLastSentPacketID(hSimConnect, &dwLastID);
```

Working Sample

[Primary sample](#) [Tracking Errors](#)

Remarks

This function should be used in conjunction with returned structures of type [SIMCONNECT_RECV_EXCEPTION](#) to help pinpoint errors (exceptions) returned by the server. This is done by matching the send ID returned with the exception, with the number returned by this function and stored appropriately. This function is primarily intended to be used while debugging and testing the client application, rather than in a final retail build.

See Also

- [SimConnect API Reference](#)

SimConnect_GetNextDispatch

The **SimConnect_GetNextDispatch** function is used to process the next SimConnect message received, without the use of a callback function.

Syntax

```
HRESULT SimConnect_GetNextDispatch(
    HANDLE hSimConnect,
    SIMCONNECT_RECV** ppData,
    DWORD* pcbData
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ppData

[in] Pointer to a pointer to a data buffer, initially to be treated as a [SIMCONNECT_RECV](#) structure. If you are going to make a copy of the data buffer (which is maintained by the SimConnect client library) make sure that the defined buffer is large enough (the size of the returned data structure is one member of the [SIMCONNECT_RECV](#) structure).

pcbData

[in] Pointer to the size of the data buffer, in bytes.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```

SIMCONNECT_RECV* pData;
DWORD cbData;

hr = SimConnect_GetNextDispatch(hSimConnect, &pData, &cbData);
if (SUCCEEDED(hr))
{
    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_OPEN:
            // enter code to handle SimConnect version information received in a
            SIMCONNECT_RECV_OPEN structure.
            SIMCONNECT_RECV_OPEN *openData = (SIMCONNECT_RECV_OPEN*) pData;
            break;

        case SIMCONNECT_RECV_ID_EVENT:
            // enter code to handle events received in a SIMCONNECT_RECV_EVENT structure.
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*) pData;
            break;

        case SIMCONNECT_RECV_ID_EVENT_FILENAME:
            // enter code to handle event filenames received in a
            SIMCONNECT_RECV_EVENT_FILENAME
            // structure.
            SIMCONNECT_RECV_EVENT_FILENAME *evt =
            (SIMCONNECT_RECV_EVENT_FILENAME*) pData;
            break;

        case SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE:
            // enter code to handle AI objects that have been added or removed, and received in a
            SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE
            // structure.
            SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE *evt =
            (SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE*) pData;
            break;

        case SIMCONNECT_RECV_ID_EVENT_FRAME:
            // enter code to handle frame data received in a SIMCONNECT_RECV_EVENT_FRAME
            // structure.
            SIMCONNECT_RECV_EVENT_FRAME *evt = (SIMCONNECT_RECV_EVENT_FRAME*)
pData;
            break;

        case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:
            // enter code to handle object data received in a SIMCONNECT_RECV_SIMOBJECT_DATA
            // structure.
            SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData =
            (SIMCONNECT_RECV_SIMOBJECT_DATA*) pData;
            break;

        case SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE:
            // enter code to handle object data received in a
            SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE
            // structure.
            SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE *pObjData =
            (SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE*) pData;
            break;

        case SIMCONNECT_RECV_ID_QUIT:
            // enter code to handle exiting the application
            break;

        case SIMCONNECT_RECV_ID_EXCEPTION:
            // enter code to handle errors received in a SIMCONNECT_RECV_EXCEPTION structure.
            SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)
    }
}

```

```

pData;
break;

case SIMCONNECT_RECV_ID_WEATHER_OBSERVATION:
    // enter code to handle object data received in a
SIMCONNECT_RECV_WEATHER_OBSERVATION structure.
    SIMCONNECT_RECV_WEATHER_OBSERVATION* pWxData =
(SIMCONNECT_RECV_WEATHER_OBSERVATION*) pData;
    const char* pszMETAR = (const char*) (pWxData+1);
    break;

    // Enter similar case statements to handle the other types of data that can be received, including:
    // SIMCONNECT_RECV_ID_ASSIGNED_OBJECT_ID,
    // SIMCONNECT_RECV_ID_RESERVED_KEY,
    // SIMCONNECT_RECV_ID_CUSTOM_ACTION
    // SIMCONNECT_RECV_ID_SYSTEM_STATE
    // SIMCONNECT_RECV_ID_CLOUD_STATE

default:
    // enter code to handle the case where an unexpected message is received
    break;
}
}

```

Working Sample

[Primary sample No Callback](#)

Remarks

It is important to call this function sufficiently frequently that the queue of information received from the server is processed. If there are no messages in the queue, the **dwID** parameter will be set to **SIMCONNECT_RECV_ID_NULL**.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CallDispatch](#)
-

SimConnect_MapClientDataNameToID

The **SimConnect_MapClientDataNameToID** function is used to associate an ID with a named client data area.

Syntax

```

HRESULT SimConnect_MapClientDataNameToID(
    HANDLE hSimConnect,
    const char* szClientDataName,
    SIMCONNECT_CLIENT_DATA_ID ClientDataID
);

```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szClientDataName

[in] Null-terminated string containing the client data area name. This is the name that another client will use to specify the data area. The name is not case-sensitive. If the name requested is already in use by another add-on, a **SIMCONNECT_EXCEPTION_ALREADY_CREATED** error will be returned.

ClientDataID

[in] A unique ID for the client data area, specified by the client. If the ID number is already in use, the **SIMCONNECT_EXCEPTION_DUPLICATE_ID** error will be returned.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the

following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

This function should be called once for each client data area: the client setting up the data should call it just before a call to [SimConnect_CreateClientData](#), and the clients requesting the data should call it before any calls to [SimConnect_RequestClientData](#) are made. The name given to a client data area must be unique, however by mapping an ID number to the name, calls to the functions to set and request the data are made more efficient.

See Also

- [SimConnect_AddToClientDataDefinition](#)
- [SimConnect_ClearClientDataDefinition](#)
- [SimConnect_CreateClientData](#)
- [SimConnect_RequestClientData](#)
- [SimConnect_SetClientData](#)
- [SimConnect API Reference](#)

SimConnect_MapClientEventToSimEvent

The **SimConnect_MapClientEventToSimEvent** function associates a client defined event ID with a *Prepar3D* event name.

Syntax

```
HRESULT SimConnect_MapClientEventToSimEvent(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_EVENT_ID EventID,
    const char* EventName
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

EventID

[in] Specifies the ID of the client event.

EventName

[in] Specifies the *Prepar3D* event name. Refer to the [Event IDs](#) document for a list of event names (listed under **String Name**). If the event name includes one or more periods (such as "Custom.Event" in the example below) then they are custom events specified by the client, and will only be recognized by another client (and not *Prepar3D*) that has been coded to receive such events. No *Prepar3D* events include periods. If no entry is made for this parameter, the event is private to the client.

Alternatively enter a decimal number in the format "#nnnn" or a hex number in the format "#0xnnnn", where these numbers are in the range **THIRD_PARTY_EVENT_ID_MIN** and **THIRD_PARTY_EVENT_ID_MAX**, in order to receive events from third-party add-ons to *Prepar3D*.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```

static enum EVENT_ID {
    EVENT_PAUSE,
    EVENT_BRAKES,
    EVENT_CUSTOM,
    EVENT_PRIVATE,
};

// Attach the client event EVENT_BRAKES to the simulation event "brakes"
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES, "brakes");

// Attach the client event EVENT_PAUSE to the simulation event "pause_toggle"
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_PAUSE,
                                         "pause_toggle");

// Create a custom event, for use when communicating with other clients
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CUSTOM,
                                         "Custom.Event");

// Create a private event for use within this client only
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_PRIVATE);

```

Working Samples

[Client Event](#)
[Cockpit Camera](#)
[Joystick Input](#)
[Reserved Key](#)
Primary samples
[Send Event A](#)
[Send Event B](#)
[Send Event C](#)
[Set Data](#)
[Throttle Control](#)
[Tracking Errors](#)

Reference samples All but a few of the other [samples](#) implement this function.

Remarks

Client events, such as EVENT_BRAKES, must be added to a group event (to set the appropriate priority) before event notifications will be received from the SimConnect server (see the [SimConnect_AddClientEventToNotificationGroup](#) function).

See Also

- [SimConnect API Reference](#)
- [SimConnect_TransmitClientEvent](#)

SimConnect_MapInputEventToClientEvent

The **SimConnect_MapInputEventToClientEvent** function is used to connect input events (such as keystrokes, joystick or mouse movements) with the sending of appropriate event notifications.

Syntax

```

HRESULT SimConnect_MapInputEventToClientEvent(
    HANDLE hSimConnect,
    SIMCONNECT_INPUT_GROUP_ID GroupID,
    const char* pszInputDefinition,
    SIMCONNECT_CLIENT_EVENT_ID DownEventID,
    DWORD DownValue = 0,
    SIMCONNECT_CLIENT_EVENT_ID UpEventID
    =(SIMCONNECT_CLIENT_EVENT_ID)SIMCONNECT_UNUSED,
    DWORD UpValue = 0,
    BOOL bMaskable = FALSE
);

```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

GroupID

[in] Specifies the ID of the client defined input group that the input event is to be added to.

pszInputDefinition

[in] Pointer to a null-terminated string containing the definition of the input events (keyboard keys, mouse or joystick events, for example). See the Remarks and example below for a range of possibilities.

DownEventID

[in] Specifies the ID of the down, and default, event. This is the client defined event that is triggered when the input event occurs. If only an up event is required, set this to

SIMCONNECT_UNUSED.

DownValue

[in, optional] Specifies an optional numeric value, which will be returned when the down event occurs. This value is only acknowledged for keyboard and button events.

UpEventID

[in, optional] Specifies the ID of the up event. This is the client defined event that is triggered when the up action occurs.

UpValue

[in, optional] Specifies an optional numeric value, which will be returned when the up event occurs. This value is only acknowledged for keyboard and button events.

bMaskable

[in, optional] If set to true, specifies that the client will mask the event, and no other lower priority clients will receive it. The default is false.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum INPUT_ID {
    INPUT_1,
};

static enum EVENT_ID {
    EVENT_1,
    EVENT_2,
    EVENT_3
};

hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1, "parking_brakes");
// Set similar mappings for EVENT_2 and EVENT_3

// Lower case a and upper case B are hit together
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "a+B", EVENT_1);

// Ctrl, upper case A and upper case U are hit together
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "Ctrl+A+U",
EVENT_1);

// Ctrl, shift, lower case a, has been hit
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "shift+ctrl+a",
EVENT_2);

// Ctrl, shift, lower case a, will trigger an EVENT_2 when it is pressed, and an EVENT_3 when
released
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "shift+ctrl+a",
EVENT_2, 0, EVENT_3);
```

```

// The first configured button of joystick 0 is hit
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1,
"joystick:0:button:0", EVENT_2);

// The second configured button of joystick 0 is hit
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1,
"joystick:0:button:1", EVENT_3);

// The first configured joystick has had its first configured point of view (or hat) switch pressed
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "joystick:0:POV:0",
EVENT_3);

// The first configured joystick has been moved along the x axis
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "joystick:0:XAxis",
EVENT_3);

hr = SimConnect_SetInputGroupPriority(hSimConnect, INPUT_1,
SIMCONNECT_GROUP_PRIORITY_HIGHEST);

```

Working Samples

	Cockpit Camera
Primary samples	Input Event
	Joystick Input
	Throttle Control

Reference samples [Set Data](#)

Remarks

The maximum number of events that can be added to an input group is 1000.

For the keyboard the input definition can include a maximum of two modifiers (**Shift**, **Ctrl**, **Alt**) and two keys (case sensitive).

For joysticks the input definition is in the form "joystick:*n*:input[:*i*]". Where *n* is the joystick number (starting from 0), *input* is the input name, and *i* is an optional index number that might be required by the input name (joystick:0:button:0 for example). The joystick number can be retrieved through the [SimConnect_RequestJoystickDeviceInfo](#) function. The input name can be one in the following table:

Input Name	Description	Range of values
Button	One of the joystick buttons, configured from 0.	0 = up state, 1 = down state
POV	Point of view switch (often called the hat switch).	0 facing ahead 4500 forward right 9000 right 13500 rear right 18000 rear 22500 rear left 27000 left 31500 forward left
Slider	The variable position slider on the joystick.	The actual values returned can vary widely on the joystick, though the limits are 32K (pulled back to the limit) to -32K (maximum forward limit).
XAxis , YAxis or ZAxis	Movement of the joystick in the X, Y, or Z directions. For most joysticks the movement is left or right for the XAxis and forward or backward for the YAxis, with no values for the ZAxis.	The limits in the Y axis are 32K (pulled back) to -32K (pushed forward). The limits in the X axis are -32K (full left) to 32K (full right). Depending on the joystick though, the limits may be significantly less than these values.
RxAxis , RyAxis , or RzAxis	Rotation of the joystick about the X, Y, or Z axis. For most joysticks there is only rotational movement around the Z axis, with no values for the X or Y axis.	For the Z axis, the limits are -32K (rotated left to the maximum) to 32K (rotated right to the maximum). Again, actual limits depend on the joystick.

For keyboard hits, usually no further information other than the key has been pressed is necessary for the client to process the event appropriately. For joystick events, other than button events, it is also

important to know the extent of the movement (or position of the hat switch, or of the slider). This information is returned with the event in the *dwData* parameter of a [SIMCONNECT_RECV_EVENT](#) structure.

For button, hat switch, or keyboard events, one event is transmitted to the client, or two if an up event is specified, when the input event occurs. If joystick axis, rotation or slider events are requested, then an event is transmitted for these six times per second whether the joystick is actually moved or not, unless the value for these is zero, in which case events are not transmitted until the joystick is moved from this position. Joystick and keyboard events are only transmitted when a scenario is loaded, not while the user is navigating the shell of the product.

For reference, the default input mappings of joystick buttons to events is specified in the Standard.xml file in the root *Lockheed Martin\Prepar3D v5* folder. The first time *Prepar3D* is run, a subset of this file is written out to a file with the same name in the %APPDATA%\Lockheed Martin\Prepar3D v5\Controls folder, containing just the mappings that the current setup is using. Each time *Prepar3D* is run the smaller file is read in, and is the first used to find a mapping for a device. If a mapping is not found the larger file in the root folder is opened and searched (and a successful mapping added to the internal copy of the file). A generic mapping is used if no match is found. The file is written out again when the simulation is exited, obviously containing any new mappings. It is usually safe for a third-party to update the smaller version of Standard.xml with correct and appropriate mappings, as long as this does not happen while *Prepar3D* is running, as in this case the file will be over-written.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetInputGroupPriority](#)
 - [SimConnect_RemoveInputEvent](#)
 - [SimConnect_ClearInputGroup](#)
 - [SimConnect_SetInputGroupState](#)
 - [SimConnect_RequestJoystickDeviceInfo](#)
-

SimConnect_Open

The **SimConnect_Open** function is used to send a request to the *Prepar3D* server to open up communications with a new client.

Syntax

```
HRESULT SimConnect_Open(  
    HANDLE* phSimConnect,  
    LPCSTR szName,  
    HWND hWnd,  
    DWORD UserEventWin32,  
    HANDLE hEventHandle,  
    DWORD ConfigIndex  
)
```

Parameters

phSimConnect

[in] Pointer to a handle to a SimConnect object.

szName

[in] Pointer to a null-terminated string containing an appropriate name for the client program. The empty string ("") is a valid entry, NULL is not supported.

hWnd

[in] Handle to a Windows object. Set this to NULL if the handle is not being used.

UserEventWin32

[in] Code number that the client can specify. Set this to 0 if it is not being used.

hEventHandle

[in] A Windows Event handle. A client can be written to respond to Windows Events, rather than use a polling and callback system, which can be a more efficient process if the client does not have to respond very frequently to changes in data in *Prepar3D*.

ConfigIndex

[in] The configuration index. The SimConnect.cfg file can contain a number of configurations, identified in sections with the [SimConnect] or [SimConnect.N] titles. Setting this configuration index indicates which configuration settings to use for this SimConnect client. This is useful for applications that communicate with a number of different machines that are running *Prepar3D*. The default configuration index is zero (matching a [SimConnect] entry in a SimConnect.cfg file). Note the E_INVALIDARG return value.

To ensure that a configuration file is not read accidentally, for example when creating a .dll add-on that is only to work locally, enter **SIMCONNECT_OPEN_CONFIGINDEX_LOCAL** for this parameter, which will force local operation regardless of the existence of any configuration files.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.
E_INVALIDARG	A SimConnect section in the Simconnect.cfg file did not contain the config index requested in the parameters.

Example

```
HRESULT hr;
HANDLE hSimConnect = NULL;

hr = SimConnect_Open(&hSimConnect, "Your Application Name", NULL, 0, 0,
SIMCONNECT_OPEN_CONFIGINDEX_LOCAL);
```

Working Samples

Primary samples [Open and Close](#)
[Windows Event](#)

Reference samples All of the other [samples](#) implement this function.

Remarks

Most client applications will have one **SimConnect_Open** call, and one corresponding **SimConnect_Close** call. However in some applications, multiplayer in particular, multiple **SimConnect_Open** calls may be necessary, in which case an array or list of handles will need to be maintained, and closed appropriately.

A client can optionally examine the **SIMCOMMENT_RECV_OPEN** structure that is returned after a call to **SimConnect_Open**. This structure gives versioning and build information that should be useful when multiple versions of SimConnect and multiple versions of *Prepar3D* that support it, are available.

If a remote client successfully establishes a link with *Prepar3D*, but at some later time the network connection is lost, SimConnect functions will return the NTSTATUS error STATUS_REMOTE_DISCONNECT (0xC000013CL).

The **SIMCONNECT_EXCEPTION_VERSION_MISMATCH** exception will be returned if a versioning error has occurred, typically when a client built on a newer version of the SimConnect client dll attempts to work with an older version of the SimConnect server. If this exception is received the *dwIndex* parameter will contain the following:

(version minor number * 65536) + version major number
. The minor number will be 0 and the major number 10. Use the **LOWORD** and **HIBYTE** macros to easily extract the two numbers.

See Also

- [SimConnect API Reference](#)
- [SimConnect_Close](#)

SimConnect_RemoveClientEvent

The **SimConnect_RemoveClientEvent** function is used to remove a client defined event from a notification group.

Syntax

```
HRESULT SimConnect_RemoveClientEvent(
    HANDLE hSimConnect,
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID,
    SIMCONNECT_CLIENT_EVENT_ID EventID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
GroupID
[in] Specifies the ID of the client defined group.
EventID
[in] Specifies the ID of the client defined event ID that is to be removed from the group.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum EVENT_ID {
    EVENT_1,
    EVENT_2
    EVENT_3
};
static enum GROUP_ID {
    GROUP_1,
};
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_1);
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_2);
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_1, EVENT_3, TRUE);

hr = SimConnect_RemoveClientEvent(hSimConnect, GROUP_1, EVENT_2);
```

Remarks

Use this function to permanently remove the client event. There is no reliable procedure to temporarily turn off a client event.

See Also

- [SimConnect API Reference](#)
- [SimConnect_AddClientEventToNotificationGroup](#)
- [SimConnect_SetNotificationGroupPriority](#)
- [SimConnect_ClearNotificationGroup](#)

SimConnect_RemoveInputEvent

The **SimConnect_RemoveInputEvent** function is used to remove an input event from a specified input group object.

Syntax

```
HRESULT SimConnect_RemoveInputEvent(
    HANDLE hSimConnect,
    SIMCONNECT_INPUT_GROUP_ID GroupID,
    const char* pszInputDefinition
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
GroupID
[in] Specifies the ID of the client defined input group from which the event is to be removed.
pszInputDefinition
[in] Pointer to a null-terminated string containing the input definition.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum INPUT_ID {
    INPUT_1,
};

static enum EVENT_ID {
    EVENT_1,
};

hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1, "parking_brakes");
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "a+B", EVENT_1);
...
hr = SimConnect_RemoveInputEvent(hSimConnect, INPUT_1, "a+B");
```

Remarks

The input string definitions must match exactly, before anything is removed from the group definition. For example, the string definitions "**A+B**" and "**a+B**" do not match.

See Also

- [SimConnect API Reference](#)
- [SimConnect_MapInputEventToClientEvent](#)
- [SimConnect_SetInputGroupPriority](#)
- [SimConnect_ClearInputGroup](#)
- [SimConnect_SetInputGroupState](#)

SimConnect_RequestClientData

The **SimConnect_RequestClientData** function is used to request that the specified data in an area created by another client be sent to this client.

Syntax

```
HRESULT SimConnect_RequestClientData(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_DATA_ID ClientDataID,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    SIMCONNECT_CLIENT_DATA_DEFINITION_ID DefineID,
    SIMCONNECT_CLIENT_DATA_PERIOD Period =
        SIMCONNECT_CLIENT_DATA_PERIOD_ONCE,
    SIMCONNECT_CLIENT_DATA_REQUEST_FLAG Flags = 0,
    DWORD origin = 0,
    DWORD interval = 0,
    DWORD limit = 0
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ClientDataID

[in] Specifies the ID of the client data area. Before calling this function for the first time on one client area, call [SimConnect_MapClientDataNameToID](#) to map an ID to the unique client data area name. This name must match the name specified by the client creating the data area with the [SimConnect_MapClientDataNameToID](#) and [SimConnect_CreateClientData](#) functions.

RequestID

[in] Specifies the ID of the client-defined request. This is used later by the client to identify which data has been received. This value should be unique for each request, re-using a RequestID will overwrite any previous request using the same ID.

DefineID

[in] Specifies the ID of the client-defined data definition. This definition specifies the data that should be sent to the client.

Period

[in, optional] One member of the [SIMCONNECT_CLIENT_DATA_PERIOD](#) enumeration type, specifying how often the data is to be sent by the server and received by the client.

Flags

[in, optional] A DWORD containing one or more of the following values:

Flag value	Description
0	The default, data will be sent strictly according to the defined period.
SIMCONNECT_CLIENT_DATA_REQUEST_FLAG_CHANGED	Data will only be sent to the client when one or more values have changed. If this flag is the only flag set, then all the variables in a data definition will be returned if just one of the values changes.
SIMCONNECT_CLIENT_DATA_REQUEST_FLAG_TAGGED	Requested data will be sent in tagged format (datum ID/value pairs). Tagged format requires that a datum reference ID is returned along with the data value, in order that the client code is able to identify the variable. This flag is usually set in conjunction with the previous flag, but it can be used on its own to return all the values in a data definition in datum ID/value pairs. See the SIMCONNECT_RECV_CLIENT_DATA structure for more details.

origin

[in, optional] The number of Period events that should elapse before transmission of the data begins. The default is zero, which means transmissions will start immediately.

interval

[in, optional] The number of Period events that should elapse between transmissions of the data. The default is zero, which means the data is transmitted every Period.

limit

[in, optional] The number of times the data should be transmitted before this communication is ended. The default is zero, which means the data should be transmitted endlessly.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

A data definition must be specified, using the [SimConnect_AddToClientDataDefinition](#) function, before this function can be called. If the data definition exceeds the size of the client data area on the server, then the extra bytes will be filled with zeros, an error will not be returned.

The data will be returned in a [SIMCONNECT_RECV_CLIENT_DATA](#) structure.

See the remarks for [SimConnect_RequestDataOnSimObject](#), as the two functions work in a very

similar manner.

This function has been updated for the SP1a release of the SDK, expanding on its functionality.

See Also

- [SimConnect_AddToClientDataDefinition](#)
 - [SimConnect_ClearClientDataDefinition](#)
 - [SimConnect_CreateClientData](#)
 - [SimConnect_MapClientDataNameToID](#)
 - [SimConnect_SetClientData](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestJoystickDeviceInfo

The **SimConnect_RequestJoystickDeviceInfo** function is used to request information on currently connected joystick devices.

Syntax

```
HRESULT SimConnect_RequestJoystickDeviceInfo(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO](#) structure.

This function does not include mouse devices.

See Also

- [SimConnect API Reference](#)
-

SimConnect_RequestNotificationGroup

The **SimConnect_RequestNotificationGroup** function is used to request events are transmitted from a notification group, when the simulation is in Dialog Mode.

Syntax

```
HRESULT SimConnect_RequestNotificationGroup(  
    HANDLE hSimConnect,  
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID,  
    DWORD dwReserved = 0,  
    DWORD Flags = 0  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

GroupID

[in] Specifies the ID of the client defined group.

dwReserved

[in, optional] Reserved for future use.

Flags

[in, optional] Reserved for future use.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
--------------	-------------

S_OK The function succeeded.

E_FAIL The function failed.

Remarks

In this version this function has the specific purpose of enabling the sending of events, particularly joystick events, when the simulation is in Dialog Mode.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AddClientEventToNotificationGroup](#)
 - [SimConnect_SetNotificationGroupPriority](#)
 - [SimConnect_ClearNotificationGroup](#)
-

SimConnect_RequestReservedKey

The **SimConnect_RequestReservedKey** function is used to request a specific keyboard TAB-key combination applies only to this client.

Syntax

```
HRESULT SimConnect_RequestReservedKey(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_EVENT_ID EventID,
    const char* szKeyChoice1,
    const char* szKeyChoice2 = "",
    const char* szKeyChoice3 = ""
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

EventID

[in] Specifies the client defined event ID.

szKeyChoice1

[in] Null-terminated string containing the first key choice. Refer to the document [Key Strings](#) for a full list of choices that can be entered for these three parameters.

szKeyChoice2

[in, optional] Null-terminated string containing the second key choice.

szKeyChoice3

[in, optional] Null-terminated string containing the third key choice.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
--------------	-------------

S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [Reserved Key](#)

Remarks

A successful call to this function will result in a [**SIMCONNECT_RECV_RESERVED_KEY**](#) structure being returned, with the key that has been assigned to this client. The first of the three that can be assigned will be the choice, unless all three are already taken, in which case a null string will be returned.

The *szKeyChoice* parameters should be a single character (such as "A"), which is requesting that the key combination TAB-A is reserved for this client. All reserved keys are TAB-key combinations.

See Also

- [SimConnect_MenuAddItem](#)
- [SimConnect API Reference](#)

SimConnect_RequestSystemState

The **SimConnect_RequestSystemState** function is used to request information from a number of Prepar3D system components.

Syntax

```
HRESULT SimConnect_RequestSystemState(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    const char* szState
);
```

Parameters

hSimConnect
Handle to a SimConnect object.

RequestID
The client defined request ID.

szState
A null-terminated string identifying the system function. One from the following table:

String	Description
AircraftLoaded	Requests the full path name of the last loaded aircraft flight dynamics file. These files have a .AIR extension.
DialogMode	Requests whether the simulation is in Dialog mode or not. See SimConnect_SetSystemState for a description of Dialog mode.
FlightLoaded	Requests the full path name of the last loaded scenario. Scenario files have the extension .FXML.
FlightPlan	Requests the full path name of the active flight plan. An empty string will be returned if there is no active flight plan.
FullScreenMode	Requests whether the simulation is in Full Screen mode or not.
Sim	Requests the state of the simulation. If 1 is returned, the user is in control of the aircraft, if 0 is returned, the user is navigating the UI. This is the same state that notifications can be subscribed to with the " SimStart " and " SimStop " string with the SimConnect_SubscribeToSystemEvent function.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_SYSTEM_STATE](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetSystemState](#)
-

SimConnect_RequestSystemStateW

The **SimConnect_RequestSystemStateW** function is used to request information from a number of *Prepar3D* system components. This version of the function supports unicode string results.

Syntax

```
HRESULT SimConnect_RequestSystemStateW(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    const wchar_t* szState
);
```

Parameters

hSimConnect

Handle to a SimConnect object.

RequestID

The client defined request ID.

szState

A null-terminated string identifying the system function

See [SimConnect_RequestSystemStateW](#) for

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_SYSTEM_STATE_W](#) structure. See [SimConnect_SubscribeToSystemEvent](#) for description of other parameters.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetSystemState](#)
-

SimConnect_RequestVersion

The **SimConnect_RequestVersion** function is used to request license type, *Prepar3D* version, and *Prepar3D* SimConnect version.

Syntax

```
HRESULT SimConnect_RequestVersion()
```

```
HANDLE hSimConnect,
SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_VERSION](#) structure.

See Also

- [SimConnect API Reference](#)
-

SimConnect_RequestSessionDuration

The **SimConnect_RequestSessionDuration** function is used to request the simulated time in seconds since the last scenario load. When in a scenario, the duration is accumulated between scenario saves/loads, such as saving and loading checkpoints.

```
HRESULT SimConnect_RequestSessionDuration(
HANDLE hSimConnect,
SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_SESSION_DURATION](#) structure.

See Also

- [SimConnect API Reference](#)
-

SimConnect_SetClientData

The **SimConnect_SetClientData** function is used to write one or more units of data to a client data area.

Syntax

```
HRESULT SimConnect_SetClientData(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_DATA_ID ClientDataID,
    SIMCONNECT_CLIENT_DATA_DEFINITION_ID DefineID,
    DWORD Flags,
    DWORD dwReserved,
    DWORD cbUnitSize,
    void* pDataSet
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

ClientDataID
[in] Specifies the ID of the client data area.

DefineID
[in] Specifies the ID of the client defined client data definition.

Flags
[in] Null, or one or more of the following flags.

Flag	Description
------	-------------

SIMCONNECT_CLIENT_DATA_SET_FLAG_TAGGED Refer to the *pDataSet* parameter and
for more details on the tagged format.

dwReserved

[in] Reserved for future use. Set to zero.

cbUnitSize

[in] Specifies the size of the data set in bytes. The server will check that this size matches exactly the size of the data definition provided in the *DefineID* parameter. An exception will be returned if this is not the case.

pDataSet

[in] Pointer to the data that is to be written. If the data is not in tagged format, this should point to the block of client data. If the data is in tagged format this should point to the first tag name (*datumID*), which is always four bytes long, which should be followed by the data itself. Any number of tag name/value pairs can be specified this way, the server will use the *cbUnitSize* parameter to determine how much data has been sent.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

A data definition must be specified, using the [SimConnect_AddToClientDataDefinition](#) function, before data can be set.

See Also

- [SimConnect_AddToClientDataDefinition](#)
 - [SimConnect_ClearClientDataDefinition](#)
-

-
- [SimConnect_CreateClientData](#)
 - [SimConnect_MapClientDataNameToID](#)
 - [SimConnect_RequestClientData](#)
 - [SimConnect API Reference](#)
-

SimConnect_SetInputGroupPriority

The **SimConnect_SetInputGroupPriority** function is used to set the priority for a specified input group object.

Syntax

```
HRESULT SimConnect_SetInputGroupPriority(  
    HANDLE hSimConnect,  
    SIMCONNECT_INPUT_GROUP_ID GroupID,  
    DWORD uPriority  
)
```

Parameters

hSimConnect
 [in] Handle to a SimConnect object.
GroupID
 [in] Specifies the ID of the client defined input group that the priority setting is to apply to.
uPriority
 [in] Specifies the priority setting for the input group. See the explanation of [SimConnect Priorities](#).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
hr=SimConnect_SetInputGroupPriority(hSimConnect,INPUT0,SIMCONNECT_GROUP_PRIORITY_HIGHEST);
```

Working Samples

[Input Event](#)

Primary samples [Joystick Input](#)

Remarks

A priority setting must be made for all input groups, otherwise event notifications will not be sent by the SimConnect server.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_MapInputEventToClientEvent](#)
 - [SimConnect_RemoveInputEvent](#)
 - [SimConnect_ClearInputGroup](#)
 - [SimConnect_SetInputGroupState](#)
-

SimConnect_SetInputGroupState

The **SimConnect_SetInputGroupState** function is used to turn requests for input event information from the server on and off.

Syntax

```
HRESULT SimConnect_SetInputGroupState(
    HANDLE hSimConnect,
    SIMCONNECT_INPUT_GROUP_ID GroupID,
    DWORD dwState
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
GroupID
[in] Specifies the ID of the client defined input group that is to have its state changed.
dwState
[in] Double word containing the new state. One member of the [SIMCONNECT_STATE](#) enumeration type.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum INPUT_ID {
    INPUT_1,
    INPUT_2
};
static enum EVENT_ID {
    EVENT_1,
    EVENT_2
};
.....
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1, "parking_brakes");
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_1, "ctrl+U+Q", EVENT_1);
hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_1, SIMCONNECT_STATE_ON);
```

Working Samples

	Input Event
Primary samples	Joystick Input
	Throttle Control

Reference samples [Set Data](#)

Remarks

The default state for input groups is to be inactive, so make sure to call this function each time an input group is to become active.

See Also

- [SimConnect API Reference](#)
- [SimConnect_MapInputEventToClientEvent](#)
- [SimConnect_SetInputGroupPriority](#)
- [SimConnect_RemoveInputEvent](#)
- [SimConnect_ClearInputGroup](#)

SimConnect_SetNotificationGroupPriority

The **SimConnect_SetNotificationGroupPriority** function is used to set the priority for a notification group.

Syntax

```
HRESULT SimConnect_SetNotificationGroupPriority(
    HANDLE hSimConnect,
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID,
    DWORD uPriority
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

GroupID

[in] Specifies the ID of the client defined group.

uPriority

[in] Requests the group's priority. See the explanation of [SimConnect Priorities](#).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
hr=SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);
```

Working Samples

[Client Event](#)
[Send Event A](#)
Primary samples [Send Event B](#)
[Send Event C](#)
[Tracking Errors](#)

Remarks

SimConnect Priorities

SimConnect constants define the following priorities:

Priority	Value	Description
SIMCONNECT_GROUP_PRIORITY_HIGHEST	1	The highest priority.
SIMCONNECT_GROUP_PRIORITY_HIGHEST_MASKABLE	10000000	The highest priority that allows events to be masked.
SIMCONNECT_GROUP_PRIORITY_STANDARD	1900000000	The standard priority.
SIMCONNECT_GROUP_PRIORITY_DEFAULT	2000000000	The default priority.
SIMCONNECT_GROUP_PRIORITY_LOWEST	4000000000	Priorities lower than this will be ignored.

Each notification group has an assigned priority, and the SimConnect server will send events out strictly in the order of priority. No two groups will be set at the same priority. If a request is received for a group to be set at a priority that has already been taken, the group will be assigned the next lowest priority that is available. This includes groups from all the clients that have opened communications with the server.

If a group has an assigned priority above

SIMCONNECT_GROUP_PRIORITY_HIGHEST_MASKABLE then it cannot mask events (hide them from other clients). If the group has a priority equal to or below

SIMCONNECT_GROUP_PRIORITY_HIGHEST_MASKABLE, then events can be masked (the maskable flag must be set by the [SimConnect_AddClientEventToNotificationGroup](#) function to do this). Note that it is possible to mask *Prepar3D* events, and therefore intercept them before they reach the simulation engine, and perhaps send new events to the simulation engine after appropriate

processing has been done. *Prepar3D*'s simulation engine is treated as SimConnect client in this regard, with a priority of **SIMCONNECT_GROUP_PRIORITY_DEFAULT**.

Input group events work in a similar manner. The priority groups are not combined though, a group and an input group can both have the same priority number. The SimConnect server manages two lists: notification groups and input groups.

A typical use of masking is to prevent *Prepar3D* itself from receiving an event, in order for the SimConnect client to completely replace the functionality in this case. Another use of masking is with co-operative clients, where there are multiple versions (perhaps a deluxe and standard version, or later and earlier versions), where the deluxe or later version might need to mask events from the other client, if they are both up and running. *Prepar3D* does not mask any events.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AddClientEventToNotificationGroup](#)
 - [SimConnect_RemoveClientEvent](#)
 - [SimConnect_ClearNotificationGroup](#)
-

SimConnect_SetSystemEventState

The **SimConnect_SetSystemEventState** function is used to turn requests for event information from the server on and off.

Syntax

```
HRESULT SimConnect_SetSystemEventState(  
    HANDLE hSimConnect,  
    SIMCONNECT_CLIENT_EVENT_ID EventID,  
    SIMCONNECT_STATE dwState  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
EventID
[in] Specifies the ID of the client event that is to have its state changed.
dwState
[in] Double word containing the state (one member of [SIMCONNECT_STATE](#)).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

See the example and working samples for [SimConnect_SubscribeToSystemEvent](#).

Remarks

If this function is not called, the default is for the state to be on. This is different from input events, which have a default state of off.

Use this function to turn system events temporarily on and off, rather than make multiple calls to [SimConnect_SubscribeToSystemEvent](#) and [SimConnect_UnsubscribeFromSystemEvent](#), which is less efficient.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_MapClientEventToSimEvent](#)
-

-
- [SimConnect_MapInputEventToClientEvent](#)
-

SimConnect_SetSystemState

The **SimConnect_SetSystemState** function is used to access a number of *Prepar3D* system components.

Syntax

```
HRESULT SimConnect_SetSystemState(  
    HANDLE hSimConnect,  
    const char* szState,  
    DWORD dwInteger,  
    float fFloat,  
    char* szString  
)
```

Parameters

hSimConnect

Handle to a SimConnect object.

szState

A null-terminated string identifying the system function. One from the following table:

String	Description
DialogMode	The <i>dwInteger</i> parameter should be set to 1 to turn Dialog mode on, or 0 to turn it off. Dialog mode enables the display of dialogs when the simulation is running in full-screen mode. When in dialog mode the 3D area of the screen will turn black, and will only revert to the simulation view when Dialog mode is turned off (not automatically when the dialog is closed). See the Working Sample for a simple example.
Sim	It is safe to set and reset Dialog mode when the simulation is being run in Windows mode.
dwInteger	It is not possible to set this state, so entering this will result in an exception being returned.
fFloat	An integer value, set depending on the value of <i>szState</i> .
szString	A float value, set depending on the value of <i>szState</i> (not currently used).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [DialogBoxMode](#)

Remarks

The integer, float and string set with this function match those in the [SIMCONNECT_RECV_SYSTEM_STATE](#) structure (which is returned if the information is requested with the [SimConnect_RequestSystemState](#) call).

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestSystemState](#)
 - [SimConnect_SubscribeToSystemEvent](#)
-

SimConnect_SubscribeToSystemEvent

The **SimConnect_SubscribeToSystemEvent** function is used to request that a specific system event is notified to the client.

Syntax

```
HRESULT SimConnect_SubscribeToSystemEvent(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_EVENT_ID EventID,
    const char* SystemEventName
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

EventID

[in] Specifies the client-defined event ID.

SystemEventName

[in] The string name for the requested system event, which should be one from the following table (note that the event names are not case-sensitive). Unless otherwise stated in the Description, notifications of the event are returned in a [SIMCONNECT_RECV_EVENT](#) structure (identify the event from the *EventID* given with this function).

General System Event Name

1sec

Request a notification every second.

4sec

Request a notification every four seconds.

6Hz

Request notifications six times per second. This is the same rate that joystick movement events are transmitted.

Request a notification when the aircraft flight dynamics file is changed.

These files have a .AIR extension. The filename is returned in a

[SIMCONNECT_RECV_EVENT_FILENAME](#) structure or a [SIMCONNECT_RECV_EVENT_FILENAME_W](#) if subscribed with the [SimConnect_SubscribeToSystemEventW](#) function.

AircraftLoaded

Crashed

Request a notification if the user aircraft crashes.

CrashReset

Request a notification when the crash cut-scene has completed.

Request a notification when a scenario is loaded. Note that when a scenario is ended, a default scenario is typically loaded, so these events will occur when scenarios are started and finished. The filename of the scenario

loaded is returned in a [SIMCONNECT_RECV_EVENT_FILENAME](#) structure or a [SIMCONNECT_RECV_EVENT_FILENAME_W](#) if subscribed with the [SimConnect_SubscribeToSystemEventW](#) function.

Request a notification when a scenario is saved correctly. The filename of the scenario saved is returned in a

[SIMCONNECT_RECV_EVENT_FILENAME](#) structure or a [SIMCONNECT_RECV_EVENT_FILENAME_W](#) if subscribed with the [SimConnect_SubscribeToSystemEventW](#) function.

Request a notification when a new flight plan is activated. The filename of the activated flight plan is returned in a

[SIMCONNECT_RECV_EVENT_FILENAME](#) structure or a [SIMCONNECT_RECV_EVENT_FILENAME_W](#) if subscribed with the [SimConnect_SubscribeToSystemEventW](#) function.

FlightPlanActivated

FlightPlanDeactivated

Request a notification when the active flight plan is de-activated.

Request notifications every visual frame. Information is returned in a [SIMCONNECT_RECV_EVENT_FRAME](#) structure.

Frame

Request notifications when the scenario is paused or unpause, and also immediately returns the current pause state (1 = paused or 0 = unpause). The state is returned in the *dwData* parameter.

Pause

Request a notification when the scenario is paused.

Request notifications for every visual frame that the simulation is paused. Information is returned in a [SIMCONNECT_RECV_EVENT_FRAME](#) structure.

PauseFrame

Request a notification when the user changes the position of their aircraft through a dialog.

PositionChanged

Request notifications when the scenario is running or not, and also

immediately returns the current state (1 = running or 0 = not running). The

Sim

	state is returned in the <i>dwData</i> parameter.
SimStart	The simulator is running. Typically the user is actively controlling the vehicle which is on the ground, underwater or in the air.
SimStop	The simulator is not running. Typically the user is loading a scenario, navigating the user interface or in a dialog.
Sound	Requests a notification when the master sound switch is changed. This request will also return the current state of the master sound switch immediately. A flag is returned in the <i>dwData</i> parameter, 0 if the switch is off, SIMCONNECT_SOUND_SYSTEM_EVENT_DATA_MASTER (0x1) if the switch is on.
Unpaused	Request a notification when the scenario is un-paused.
View	Requests a notification when the user aircraft view is changed. This request will also return the current view immediately. A flag is returned in the <i>dwData</i> parameter, one of: SIMCONNECT_VIEW_SYSTEM_EVENT_DATA_COCKPIT_2D SIMCONNECT_VIEW_SYSTEM_EVENT_DATA_COCKPIT_VIRTUAL SIMCONNECT_VIEW_SYSTEM_EVENT_DATA_ORTHOGONAL (the map view).
WeatherModeChanged	Request a notification when the weather mode is changed.
TextEventCreated	Request a notification when a SimConnect message is sent using SimConnect_Text . Refer also to the SIMCONNECT_RECV_EVENT_TEXT structure.
TextEventDestroyed	Request a notification when a SimConnect text window is being destroyed. Refer also to the SIMCONNECT_RECV_EVENT_TEXT_DESTROYED structure.
AI Specific Event Name	
ObjectAdded	Request a notification when an AI object is added to the simulation. Refer also to the SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE structure.
ObjectRemoved	Request a notification when an AI object is removed from the simulation. Refer also to the SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE structure.
Mission Specific Event Name	
MissionCompleted	Request a notification when the user has completed a scenario. Refer also to the SIMCONNECT_MISSION_END enum.
CustomMissionActionExecuted	Request a notification when a scenario action has been executed. Refer also to the SimConnect_CompleteCustomMissionAction function.
FlightSegmentReadyForGrading	Request a notification when a Flight Segment has been completed and is ready for grading.
Multiplayer Racing Event Name	
MultiplayerClientStarted	Used by a client to request a notification that they have successfully joined a multiplayer race. The event is returned as a SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED structure. This event is only sent to the client, not the host of the session.
MultiplayerServerStarted	Used by a host of a multiplayer race to request a notification when the race is open to other players in the lobby. The event is returned in a SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED structure.
MultiplayerSessionEnded	Request a notification when the multiplayer race session is terminated. The event is returned in a SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED structure. If a client player leaves a race, this event will be returned just to the client. If a host leaves or terminates the session, then all players will receive this event. This is the only event that will be broadcast to all players.
RaceEnd	Request a notification of the race results for each racer. The results will be returned in SIMCONNECT_RECV_EVENT_RACE_END structures, one for each player.
RaceLap	Request a notification of the race results for each racer. The results will be returned in SIMCONNECT_RECV_EVENT_RACE_LAP structures, one for each player.
Recorder Specific Event Name	
PlaybackStateChanged	Request a notification when the state of Prepar3D playback changes. Either playback has started or ended. Information is returned in a SIMCONNECT_RECV_PLAYBACK_STATE_CHANGED structure.
RecorderStateChanged	Request a notification when the state of Prepar3D recording changes. Either recording has started or ended. Information is returned in a SIMCONNECT_RECV_RECORDER_STATE_CHANGED structure.

Weapon System Event Name	PROFESSIONAL PLUS ONLY
WeaponFired	Request a notification when a detachable weapon is fired. Refer also to the SIMCONNECT_RECV_EVENT_WEAPON structure.
WeaponDetonated	Request a notification when a detachable weapon is detonated or destroyed. Refer also to the SIMCONNECT_RECV_EVENT_WEAPON structure.
CountermeasureFired	Request a notification when a countermeasure is deployed. Refer also to the SIMCONNECT_RECV_EVENT_COUNTERMEASURE structure.
ObjectDamagedByWeapon	Request a notification when an object is damaged by a weapon. Refer also to the SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON structure.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum EVENT_ID {
    EVENT_FLIGHT_LOAD,
    EVENT_RECUR_1SEC,
    EVENT_RECUR_FRAME,
};

hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_FLIGHT_LOAD,
"FlightLoaded");
hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_RECUR_1SEC, "1sec");
hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_RECUR_FRAME,
"frame");

// The recurring events will be on by default, so set one of them to off.
hr = SimConnect_SetSystemEventState(hSimConnect, EVENT_RECUR_FRAME,
SIMCONNECT_STATE_OFF);
```

Working Samples

Primary samples [Mission Action](#)
[System Event](#)

Reference samples Many of the other [samples](#) implement this function.

Remarks

A single call to this function is all that is necessary to receive the notifications. For greatest efficiency use [SimConnect_SetSystemEventState](#) to turn these requests on and off temporarily, and call [SimConnect_UnsubscribeFromSystemEvent](#) once only to permanently terminate the notifications of these events.

See Also

- [SimConnect_SubscribeToSystemEventEx](#)
- [SimConnect_RequestSystemState](#)
- [SimConnect_SetSystemState](#)
- [SimConnect API Reference](#)

SimConnect_SubscribeToSystemEventEx

The [SimConnect_SubscribeToSystemEventEx](#) function is used to request that a specific system event is notified to the client. The Ex specific version of the function adds a Flags parameter.

Syntax

```
HRESULT SimConnect_SubscribeToSystemEventEx(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_EVENT_ID EventID,
    const char* SystemEventName,
    SIMCONNECT_EVENT_SUBSCRIPTION_FLAG Flags
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

EventID

[in] Specifies the client-defined event ID.

SystemEventName

[in] The string name for the requested system event, which should be one from the following table (note that the event names are not case-sensitive). Unless otherwise stated in the Description, notifications of the event are returned in a [SIMCONNECT_RECV_EVENT](#) structure (identify the event from the *EventID* given with this function).

Flags

[in] One or more of the following flags.

Flags	Description
SIMCONNECT_EVENT_SUBSCRIPTION_FLAG_BLOCK	Specifies that a blocking callback will be used when this system event occurs. See SimConnect_RequestSynchronousBlock for more information.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

See [SimConnect_SubscribeToSystemEvent](#) for description of other parameters.

See Also

- [SimConnect_SubscribeToSystemEvent](#)
- [SimConnect_RequestSystemState](#)
- [SimConnect_SetSystemState](#)
- [SimConnect API Reference](#)

SimConnect_SubscribeToSystemEventW

The **SimConnect_SubscribeToSystemEventW** function is used to request that a specific system event is notified to the client. This version of the function supports unicode string results.

Syntax

```
HRESULT SimConnect_SubscribeToSystemEventEx(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_EVENT_ID EventID,
    const wchar_t* SystemEventName,
    SIMCONNECT_EVENT_SUBSCRIPTION_FLAG Flags
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

EventID

[in] Specifies the client-defined event ID.
SystemEventName
[in] The string name for the requested system event, which should be one from the following table (note that the event names are not case-sensitive). Unless otherwise stated in the Description, notifications of the event are returned in a [SIMCONNECT_RECV_EVENT](#) structure (identify the event from the *EventID* given with this function).

Flags

[in] One or more of the following flags.

Flags	Description
SIMCONNECT_EVENT_SUBSCRIPTION_FLAG_BLOCK	Specifies that a blocking callback will be used when this system event occurs. See SimConnect_RequestSynchronousBlock for more information.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

See [SimConnect_SubscribeToSystemEvent](#) for description of other parameters.

See Also

- [SimConnect_SubscribeToSystemEvent](#)
- [SimConnect_RequestSystemState](#)
- [SimConnect_SetSystemState](#)
- [SimConnect API Reference](#)

SimConnect_TransmitClientEvent

The **SimConnect_TransmitClientEvent** function is used to request that the *Prepar3D* server transmit to all SimConnect clients the specified client event.

Syntax

```
HRESULT SimConnect_TransmitClientEvent(
    HANDLE hSimConnect,
    SIMCONNECT_OBJECT_ID ObjectID,
    SIMCONNECT_CLIENT_EVENT_ID EventID,
    DWORD dwData,
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID,
    SIMCONNECT_EVENT_FLAG Flags
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ObjectID

[in] Specifies the ID of the server defined object. If this parameter is set to **SIMCONNECT_OBJECT_ID_USER**, then the transmitted event will be sent to the other clients in priority order. If this parameter contains another object ID, then the event will be sent direct to that sim-object, and no other clients will receive it.

EventID

[in] Specifies the ID of the client event.

dwData

[in] Double word containing any additional number required by the event. This is often zero. If the event is a *Prepar3D* event, then refer to the [Event IDs](#) document for information on this additional value. If the event is a custom event, then any value put in this parameter will be available to the clients that receive the event.

GroupID

[in] The default behavior is that this specifies the GroupID of the event. The SimConnect server will use the priority of this group to send the message to all clients with a lower priority. To receive the event notification other SimConnect clients must have subscribed to receive the event. See the explanation of [SimConnect Priorities](#). The exception to the default behavior is set by the **SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY** flag, described below.

Flags

[in] One or more of the following flags:

NOTE: For panel control you must specify a group priority of 0.

Flag	Description
0	Do nothing.
SIMCONNECT_EVENT_FLAG_SLOW_REPEAT_TIMER	The flag will effectively reset the repeat timer to simulate slow repeat. Use this flag to reset the repeat timer that works with various keyboard events and mouse clicks.
SIMCONNECT_EVENT_FLAG_FAST_REPEAT_TIMER	The flag will effectively reset the repeat timer to simulate fast repeat.
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY	Indicates to the SimConnect server to treat the <i>GroupID</i> as a priority value. If this flag is set, and the <i>GroupID</i> parameter is set to SIMCONNECT_GROUP_PRIORITY_HIGHEST then all client notification groups that have subscribed to the event will receive the notification (unless one of them masks it). The event will be transmitted to clients starting at the priority specified in the <i>GroupID</i> parameter, though this can result in the client that transmitted the event, receiving it again.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Examples

```
// 1. Transmitting a custom event to other clients
static enum EVENT_ID {
    EVENT_MY_EVENT
    EVENT_DME
    EVENT_OPEN_PANEL
};
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MY_EVENT,
                                         "Custom.Event");
// Send EVENT_MY_EVENT to all current SimConnect clients.
SimConnect_TransmitClientEvent(hSimConnect, 0, EVENT_MY_EVENT, 0,
                               SIMCONNECT_GROUP_PRIORITY_HIGHEST,
                               SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

// 2. Setting an event value
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_DME, "DME_SELECT");
// Set the selected DME to 2
SimConnect_TransmitClientEvent(hSimConnect, 0, EVENT_DME, 2,
                               SIMCONNECT_GROUP_PRIORITY_DEFAULT,
                               SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
```

NOTE: For panel control you must specify a group priority of 0.

```
SimConnect_TransmitClientEvent(hSimConnect, 0, EVENT_OPEN_PANEL, 0, 0,
                               SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
```

Working Sample

Primary sample Send Event A

Remarks

Typically use this function to transmit an event to other SimConnect clients, including the simulation engine, although the client that transmits the event can also receive it. The order in which client notification groups are informed of the event is determined by the priority of each group. The higher the priority of the group, the earlier it will receive the event notification. Refer to the explanation of the **maskable** parameter for the [SimConnect_AddClientEventToNotificationGroup](#) call, which describes when the event may be masked and not transmitted to lower priority groups. Also see the explanation of [SimConnect Priorities](#).

See Also

- [SimConnect API Reference](#)
 - [SimConnect_MapClientEventToSimEvent](#)
-

SimConnect_TransmitClientEvent64

The **SimConnect_TransmitClientEvent64** function is used to request that the *Prepar3D* server transmit to all SimConnect clients the specified client event. This function behaves the same as [SimConnect_TransmitClientEvent](#) with the following exceptions:

1. This function accepts a 64-bit user context data parameter.
2. This function sends a [SIMCONNECT_RECV](#) structure with a **dwID** value of [SIMCONNECT_RECV_ID_EVENT_64](#).
3. The received [SIMCONNECT_RECV](#) structure should be casted to type [SIMCONNECT_RECV_EVENT_64](#).
4. The 64-bit **qwData** parameter of the [SIMCONNECT_RECV_EVENT_64](#) structure contains the user context data.
5. The 32-bit **dwData** parameter of the [SIMCONNECT_RECV_EVENT_64](#) structure is unused.

Syntax

```
HRESULT SimConnect_TransmitClientEvent64(
    HANDLE hSimConnect,
    SIMCONNECT_OBJECT_ID ObjectID,
    SIMCONNECT_CLIENT_EVENT_ID EventID,
    QWORD qwData,
    SIMCONNECT_NOTIFICATION_GROUP_ID GroupID,
    SIMCONNECT_EVENT_FLAG Flags
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ObjectID

[in] Specifies the ID of the server defined object. If this parameter is set to [SIMCONNECT_OBJECT_ID_USER](#), then the transmitted event will be sent to the other clients in priority order. If this parameters contains another object ID, then the event will be sent direct to that sim-object, and no other clients will receive it.

EventID

[in] Specifies the ID of the client event.

qwData

[in] Quad word containing any additional number required by the event. This is often zero. If the event is a *Prepar3D* event, then refer to the [Event IDs](#) document for information on this additional value. If the event is a custom event, then any value put in this parameter will be available to the clients that receive the event.

GroupID

[in] The default behavior is that this specifies the GroupID of the event. The SimConnect server will use the priority of this group to send the message to all clients with a lower priority. To receive the event notification other SimConnect clients must have subscribed to receive the event. See the explanation of [SimConnect Priorities](#). The exception to the default behavior is set by the [SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY](#) flag, described below.

Flags

[in] One or more of the following flags:

NOTE: For panel control you must specify a group priority of 0.

Flag	Description
0	Do nothing.
SIMCONNECT_EVENT_FLAG_SLOW_REPEAT_TIMER	The flag will effectively reset the repeat timer to simulate slow repeat. Use this flag to reset the repeat timer that works with various keyboard events and mouse clicks.
SIMCONNECT_EVENT_FLAG_FAST_REPEAT_TIMER	The flag will effectively reset the repeat timer to simulate fast repeat.
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY	Indicates to the SimConnect server to treat the <i>GroupID</i> as a priority value. If this flag is set, and the <i>GroupID</i> parameter is set to SIMCONNECT_GROUP_PRIORITY_HIGHEST then all client notification groups that have subscribed to the event will receive the notification (unless one of them masks it). The event will be transmitted to clients starting at the priority specified in the <i>GroupID</i> parameter, though this can result in the client that transmitted the event, receiving it again.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Examples

```
// 1. Transmitting a custom event to other clients
static enum EVENT_ID {
    EVENT_MY_EVENT
    EVENT_DME
    EVENT_OPEN_PANEL
};
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MY_EVENT,
    "Custom.Event");
// Send EVENT_MY_EVENT to all current SimConnect clients.
SimConnect_TransmitClientEvent64(hSimConnect, 0, EVENT_MY_EVENT, 0,
    SIMCONNECT_GROUP_PRIORITY_HIGHEST,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);

// 2. Setting an event value
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_DME, "DME_SELECT");
// Set the selected DME to 2
SimConnect_TransmitClientEvent64(hSimConnect, 0, EVENT_DME, 2,
    SIMCONNECT_GROUP_PRIORITY_DEFAULT,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
```

NOTE: For panel control you must specify a group priority of 0.

```
SimConnect_TransmitClientEvent64(hSimConnect, 0, EVENT_OPEN_PANEL, 0, 0,
    SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);
```

Remarks

Typically use this function to transmit an event to other SimConnect clients, including the simulation engine, although the client that transmits the event can also receive it. The order in which client notification groups are informed of the event is determined by the priority of each group. The higher the priority of the group, the earlier it will receive the event notification. Refer to the explanation of the

maskable parameter for the [SimConnect_AddClientEventToNotificationGroup](#) call, which describes when the event may be masked and not transmitted to lower priority groups. Also see the explanation of [SimConnect Priorities](#).

See Also

- [SimConnect API Reference](#)
 - [SimConnect_TransmitClientEvent](#)
 - [SimConnect_MapClientEventToSimEvent](#)
-

SimConnect_UnsubscribeFromSystemEvent

The **SimConnect_UnsubscribeFromSystemEvent** function is used to request that notifications are no longer received for the specified system event.

Syntax

```
HRESULT SimConnect_UnsubscribeFromSystemEvent(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_EVENT_ID EventID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
EventID
[in] Specifies the client-defined event ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum EVENT_ID {
    EVENT_FLIGHT_LOAD,
    EVENT_RECUR_1SEC,
    EVENT_RECUR_FRAME,
};

hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_RECUR_1SEC);
...
hr = SimConnect_UnsubscribeFromSystemEvent(hSimConnect, EVENT_RECUR_1SEC);
```

Remarks

There is no limit to the number of system events that can be subscribed to, but use this function to improve performance when a system event notification is no longer needed.

See Also

- [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_InsertString

The **SimConnect_InsertString** function is used to assist in adding variable length narrow strings to a structure.

Syntax

```
HRESULT SimConnect_InsertString(
    BYTE* pDest,
    DWORD cbDest,
    void** ppEnd,
    DWORD* pcbStringV,
    const char* pSource
);
```

Parameters

pDest

[in] Pointer to where the source string is to be written in the destination object.

cbDest

[in] The size of the remaining space in the destination object.

ppEnd

[in,out] Pointer to a pointer, (usually a pointer to a char pointer). On return the pointer locates the end of the string in the structure, and hence the starting position for any other string to be included in the structure.

pcbStringV

[in,out] Pointer to a DWORD. On returning this DWORD will contain the size of the source string in bytes.

pSource

[in] Pointer to the source string.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

This function does not communicate with the SimConnect server, but is a helper function to assist in the handling of variable length narrow strings. Its counterpart is the [SimConnect_RetrieveString](#) function.

See Also

- [SimConnect API Reference](#)
- [SimConnect_RetrieveString](#)

SimConnect_InsertStringW

The **SimConnect_InsertStringW** function is used to assist in adding variable length wide strings to a structure.

Syntax

```
HRESULT SimConnect_InsertStringW(
    BYTE* pDest,
    DWORD cbDest,
    void** ppEnd,
    DWORD* pcbStringV,
    const wchar_t* pSource
);
```

Parameters

pDest

[in] Pointer to where the source string is to be written in the destination object.

cbDest

[in] The size of the remaining space in the destination object.

ppEnd
[in,out] Pointer to a pointer, (usually a pointer to a wchar_t pointer). On return the pointer locates the end of the string in the structure, and hence the starting position for any other string to be included in the structure.

pcbStringV
[in,out] Pointer to a DWORD. On returning this DWORD will contain the size of the source string in bytes.

pSource
[in] Pointer to the source string.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

This function does not communicate with the SimConnect server, but is a helper function to assist in the handling of variable length wide strings. Its counterpart is the [SimConnect_RetrieveStringW](#) function.

See Also

- [SimConnect API Reference](#)
- [SimConnect_RetrieveStringW](#)

SimConnect_RequestResponseTimes

The **SimConnect_RequestResponseTimes** function is used to provide some data on the performance of the client-server connection.

Syntax

```
HRESULT SimConnect_RequestResponseTimes(
    HANDLE hSimConnect,
    DWORD nCount,
    float* fElapsedSeconds
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

nCount
[in] Integer containing the number of elements in the array of floats. This should be set to five for the full range of timings, but can be less if only the first few are of interest. There is no point creating an array of greater than five floats.

fElapsedSeconds
[in] An array of *nCount* floats, containing the times. The five elements will contain the following: 0 - total round trip time, 1 - time from the request till the packet is sent, 2 - time from the request till the packet is received by the server, 3 - time from the request till the response is made by the server, 4 - time from the server response to the client receives the packet.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
int quit = 0;
bool fTesting = true;
....
while( quit == 0 )
{
    hr = SimConnect_CallDispatch(hSimConnect, MyDispatchProc, NULL);
    Sleep(0);
    if (fTesting)
    {
        fTesting = false;
        float fElapsedSeconds[5];
        hr = SimConnect_RequestResponseTimes(hSimConnect, 5, &fElapsedSeconds[0]);
        // Enter code to display the contents of fElapsedSeconds
    }
}
```

Remarks

This function should not be used as part of a final application, as it is costly in performance, but is available to help provide some performance data that can be used while building and testing a client application.

See Also

- [SimConnect API Reference](#)
-

SimConnect_RetrieveString

The **SimConnect_RetrieveString** function is used to assist in retrieving variable length narrow strings from a structure.

Syntax

```
HRESULT SimConnect_RetrieveString(
    SIMCONNECT_RECV* pData,
    DWORD cbData,
    void* pStringV,
    char** ppszString,
    DWORD* pcbString
);
```

Parameters

pData

[in] Pointer to a **SIMCONNECT_RECV** structure, containing the data.

cbData

[in] The size of the structure that inherits the **SIMCONNECT_RECV** structure, in bytes.

pStringV

[in] Pointer to the start of the variable length string within the structure.

ppszString

[in, out] Specifies a pointer to a pointer to a character buffer that should be large enough to contain the maximum length of string that might be returned. On return this buffer should contain the retrieved string.

pcbString

[in, out] Pointer to a DWORD. On return this contains the length of the string in bytes.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
struct StructVS {
    char title[1];
}
StructVS *pS = (StructVS*)&pObjData->dwData;
char *psztitle;
DWORD cbtitle;
hr = SimConnect_RetrieveString(pData, cbData, &pS->strings, &psztitle, &cbtitle))
```

Working Sample

Primary sample [Variable Strings](#)

Remarks

This function does not communicate with the SimConnect server, but is a helper function to assist in the handling of variable length narrow strings. Its counterpart is the [SimConnect_InsertString](#) function. Note that this function works in the case where an empty string is in the structure returned by the server.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_InsertString](#)
-

SimConnect_RetrieveStringW

The **SimConnect_RetrieveStringW** function is used to assist in retrieving variable length wide strings from a structure.

Syntax

```
HRESULT SimConnect_RetrieveStringW(
    SIMCONNECT_RECV* pData,
    DWORD cbData,
    void* pStringV,
    wchar_t** ppszString,
    DWORD* pcbString
);
```

Parameters

pData

[in] Pointer to a **SIMCONNECT_RECV** structure, containing the data.

cbData

[in] The size of the structure that inherits the **SIMCONNECT_RECV** structure, in bytes.

pStringV

[in] Pointer to a the start of the variable length string within the structure.

ppszString

[in, out] Specifies a pointer to a pointer to a character buffer that should be large enough to contain the maximum length of string that might be returned. On return this buffer should contain the retrieved string.

pcbString

[in, out] Pointer to a DWORD. On return this contains the length of the string in bytes.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
struct StructVS {
    char title[1];
}
StructVS *pS = (StructVS*)&pObjData->dwData;
wchar_t *psztitle;
DWORD cbtitle;
hr = SimConnect_RetrieveStringW(pData, cbData, &pS->strings, &psztitle, &cbtitle)))
```

Working Sample

Primary sample [Variable Strings](#)

Remarks

This function does not communicate with the SimConnect server, but is a helper function to assist in the handling of variable length wide strings. Its counterpart is the [SimConnect_InsertStringW](#) function. Note that this function works in the case where an empty string is in the structure returned by the server.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_InsertStringW](#)
-

Synchronous SimConnect Specific Functions

Overview

The Synchronous SimConnect API functions are used to synchronize the code running in your SimConnect add-on with the main sim. The Synchronous APIs do this by "pausing" execution of the sim code until it receives a release command from the SimConnect client, or a timeout occurs (default timeout value is 1.5 seconds).

The Synchronous API has 3 specific APIs that allow creating just Synchronous blocks and releases.

There are also extensions to other APIs related to Synchronous SimConnect. The [External Sim API](#) uses the Synchronous SimConnect APIs implicitly. The [requestData/SetData](#), [RequestClientData/SetClientData](#), and [RequestGroundInfo](#) APIs have new _FLAG_BLOCK/_FLAG_UNBLOCK flag values to block/unblock the sim side code. There's also an extended [SubscribeToSystemEventEx](#) that includes a new flags value with a _FLAG_BLOCK.

The Synchronous API maintains a timeout check and if your add-on code hasn't released the block within the timeout value, then the block is released automatically and an exception is sent to your add-on code. The default value for this timeout is 1.5 seconds.

SimConnect_RequestSynchronousBlock

The **SimConnect_RequestSynchronousBlock** function is used to request a recurring callback (based on the *Period*) that is also a blocking callback.

Syntax

```
HRESULT SimConnect_RequestSynchronousBlock(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    SIMCONNECT_PERIOD Period,
    SIMCONNECT_DATA_REQUEST_FLAG Flags = 0,
    DWORD origin = 0,
    DWORD interval = 0,
    DWORD limit = 0
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID
[in] Specifies the ID of the client defined request. This is used later by the client to identify which request is receiving a reply. This value should be unique, re-using a *RequestID* will overwrite any previous request using the same ID.

Period
[in] One member of the [SIMCONNECT_PERIOD](#) enumeration type, specifying how often the reply is to be sent by the server and received by the client.

Flags
[in, optional] A DWORD containing one or more of the following values:

Flag value	Description
0	The default, will assume SIMCONNECT_DATA_REQUEST_FLAG_BLOCK was provided.
SIMCONNECT_DATA_REQUEST_FLAG_BLOCK	Requested reply will be sent using a blocking callback.

origin

[in, optional] The number of *Period* events that should elapse before transmission of the data begins. The default is zero, which means transmissions will start immediately.

interval

[in, optional] The number of *Period* events that should elapse between transmissions of the data. The default is zero, which means the data is transmitted every *Period*.

limit

[in, optional] The number of times the data should be transmitted before this communication is ended. The default is zero, which means the data should be transmitted endlessly.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_SynchronousUnblock](#)
- [SimConnect_SetSynchronousTimeout](#)
- [SIMCONNECT_PERIOD](#)
- [SimConnect API Reference](#)

SimConnect_SynchronousUnblock

The **SimConnect_SynchronousUnblock** function is used to release the block on any blocking callback.

Syntax

```
HRESULT SimConnect_SynchronousUnblock(  
    HANDLE hSimConnect  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [External Sim](#)

See Also

- [SimConnect_RequestSynchronousBlock](#)
 - [SimConnect_SetSynchronousTimeout](#)
 - [SimConnect API Reference](#)
-

SimConnect_SetSynchronousTimeout

The **SimConnect_SetSynchronousTimeout** function is used to set the timeout value for any blocking callbacks (if this time is exceeded without the block being released, the block is released and an exception is sent to the client).

Syntax

```
HRESULT SimConnect_SetSynchronousTimeout(
    HANDLE hSimConnect
    float fTimeSeconds
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
fTimeSeconds
[in] New timeout value for blocking callbacks.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_RequestSynchronousBlock](#)
 - [SimConnect_SynchronousUnblock](#)
 - [SimConnect API Reference](#)
-

Menu Specific Functions

SimConnect_MenuAddItem

The **SimConnect_MenuAddItem** function is used to add a menu item associated with a client event.

Syntax

```
HRESULT SimConnect_MenuAddItem(
    HANDLE hSimConnect,
```

```
const char* szMenuItem,
SIMCONNECT_CLIENT_EVENT_ID MenuEventID,
DWORD dwData
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szMenuItem

[in] Null-terminated string containing the text for the menu item.

MenuEventID

[in] Specifies the client defined event ID, which is to be transmitted when the menu item is selected (in the *uEventID* parameter of the [SIMCONNECT_RECV_EVENT](#) structure).

dwData

[in] Double word containing a data value that the client can specify for its own use (it will be returned in the *dwData* parameter of the [SIMCONNECT_RECV_EVENT](#) structure).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [Menu Items](#)

Remarks

The menu item will be added to the *Add-ons* menu. The *Add-ons* menu will only appear if there is at least one menu entry. Sub-menu items can be associated with this menu item, see

[SimConnect_MenuAddSubItem](#). If the text for the menu item should change, then remove the menu item first before adding the menu item with the correct text (see [SimConnect_MenuDeleteItem](#)).

Each client can add a number of main menu items.

See Also

- [SimConnect_MenuDeleteItem](#)
- [SimConnect_MenuDeleteSubItem](#)
- [SimConnect API Reference](#)

SimConnect_MenuAddItem64

The **SimConnect_MenuAddItem64** function is used to add a menu item associated with a client event. This function behaves the same as [SimConnect_MenuAddItem](#) with the following exceptions:

1. This function accepts a 64-bit user context data parameter.
2. This function sends a [SIMCONNECT_RECV](#) structure with a **dwID** value of **SIMCONNECT_RECV_ID_EVENT_64**.
3. The received [SIMCONNECT_RECV](#) structure should be casted to type **SIMCONNECT_RECV_EVENT_64**
4. The 64-bit **qwData** parameter of the **SIMCONNECT_RECV_EVENT_64** structure contains the user context data.
5. The 32-bit **dwData** parameter of the **SIMCONNECT_RECV_EVENT_64** structure is unused.

Syntax

```
HRESULT SimConnect_MenuAddItem64(
HANDLE hSimConnect,
const char* szMenuItem,
SIMCONNECT_CLIENT_EVENT_ID MenuEventID,
```

```
QWORD qwData  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szMenuItem

[in] Null-terminated string containing the text for the menu item.

MenuEventID

[in] Specifies the client defined event ID, which is to be transmitted when the menu item is selected (in the *uEventID* parameter of the [SIMCONNECT_RECV_EVENT_64](#) structure).

qwData

[in] Quad word containing a data value that the client can specify for its own use (it will be returned in the *qwData* parameter of the [SIMCONNECT_RECV_EVENT_64](#) structure).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

[Primary sample](#) [Menu Items](#)

Remarks

The menu item will be added to the *Add-ons* menu. The *Add-ons* menu will only appear if there is at least one menu entry. Sub-menu items can be associated with this menu item, see [SimConnect_MenuAddSubItem64](#). If the text for the menu item should change, then remove the menu item first before adding the menu item with the correct text (see [SimConnect_MenuDeleteItem](#)).

Each client can add a number of main menu items.

See Also

- [SimConnect_MenuDeleteItem](#)
- [SimConnect_MenuDeleteSubItem](#)
- [SimConnect API Reference](#)

[SimConnect_MenuAddSubItem](#)

The **SimConnect_MenuAddSubItem** function is used to add a sub-menu item associated with a client event.

Syntax

```
HRESULT SimConnect_MenuAddSubItem(  
    HANDLE hSimConnect,  
    SIMCONNECT_CLIENT_EVENT_ID MenuEventID,  
    const char* szMenuItem,  
    SIMCONNECT_CLIENT_EVENT_ID SubMenuEventID,  
    DWORD dwData  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

EventID

[in] Specifies the client defined menu event ID. This is the ID of the menu item that this item should be added to.

szMenuItem
[in] Null-terminated string containing the text for the sub-menu item.

EventID
[in] Specifies the client defined sub-menu event ID, which is to be transmitted when the sub-menu item is selected (in the *uEventID* parameter of the [SIMCONNECT_RECV_EVENT](#) structure).
This ID must be unique across all sub-menu items for the client.

dwData
[in] Double word containing a data value that the client can specify for its own use (it will be returned in the *dwData* parameter of the [SIMCONNECT_RECV_EVENT](#) structure).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

A maximum of 16 sub-menu items may be added to any one main menu item. Sub-menu items are always added to the end of the sub-menu item list. An exception, **SIMCONNECT_EXCEPTION_TOO_MANY_OBJECTS**, will be returned if an attempt is made to add more than 16 sub-menu items.

See Also

- [SimConnect_MenuAddItem](#)
- [SimConnect_MenuDeleteItem](#)
- [SimConnect_MenuDeleteSubItem](#)
- [SimConnect API Reference](#)

SimConnect_MenuAddSubItem64

The **SimConnect_MenuAddSubItem64** function is used to add a sub-menu item associated with a client event. This function behaves the same as [SimConnect_MenuAddSubItem](#) with the following exceptions:

1. This function accepts a 64-bit user context data parameter.
2. This function sends a [SIMCONNECT_RECV](#) structure with a **dwID** value of **SIMCONNECT_RECV_ID_EVENT_64**.
3. The received [SIMCONNECT_RECV](#) structure should be casted to type [SIMCONNECT_RECV_EVENT_64](#)
4. The 64-bit **qwData** parameter of the [SIMCONNECT_RECV_EVENT_64](#) structure contains the user context data.
5. The 32-bit **dwData** parameter of the [SIMCONNECT_RECV_EVENT_64](#) structure is unused.

Syntax

```
HRESULT SimConnect_MenuAddSubItem64(
    HANDLE hSimConnect,
    SIMCONNECT_CLIENT_EVENT_ID MenuEventID,
    const char* szMenuItem,
    SIMCONNECT_CLIENT_EVENT_ID SubMenuEventID,
    QWORD qwData
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

EventID
[in] Specifies the client defined menu event ID. This is the ID of the menu item that this item should be added to.

szMenuItem

[in]	Null-terminated string containing the text for the sub-menu item.
EventID	[in] Specifies the client defined sub-menu event ID, which is to be transmitted when the sub-menu item is selected (in the <i>uEventID</i> parameter of the SIMCONNECT_RECV_EVENT_64 structure). This ID must be unique across all sub-menu items for the client.
qwData	[in] Quad word containing a data value that the client can specify for its own use (it will be returned in the <i>qwData</i> parameter of the SIMCONNECT_RECV_EVENT_64 structure).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

A maximum of 16 sub-menu items may be added to any one main menu item. Sub-menu items are always added to the end of the sub-menu item list. An exception, **SIMCONNECT_EXCEPTION_TOO_MANY_OBJECTS**, will be returned if an attempt is made to add more than 16 sub-menu items.

See Also

- [SimConnect_MenuAddItem64](#)
 - [SimConnect_MenuDeleteItem](#)
 - [SimConnect_MenuDeleteSubItem](#)
 - [SimConnect API Reference](#)
-

SimConnect_MenuDeleteItem

The **SimConnect_MenuDeleteItem** function is used to remove a client defined menu item.

Syntax

```
HRESULT SimConnect_MenuDeleteItem(
    HANDLE hSimConnect,
    const SIMCONNECT_CLIENT_EVENT_ID MenuEventID
);
```

Parameters

hSimConnect	[in] Handle to a SimConnect object.
MenuEventID	[in] Specifies the client defined event ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

[Primary sample Menu Items](#)

Remarks

Menu items should be removed before a client closes. Removing the main menu item will remove any associated sub-menu items. Also see the remarks for [SimConnect_MenuAddItem](#).

See Also

- [SimConnect_MenuDeleteSubItem](#)
 - [SimConnect API Reference](#)
-

SimConnect_MenuDeleteSubItem

The **SimConnect_MenuDeleteSubItem** function is used to remove a specified sub-menu item.

Syntax

```
HRESULT SimConnect_MenuDeleteSubItem(  
    HANDLE hSimConnect,  
    SIMCONNECT_CLIENT_EVENT_ID MenuEventID,  
    const SIMCONNECT_CLIENT_EVENT_ID SubMenuEventID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

MenuEventID

[in] Specifies the client defined menu event ID, from which the sub-menu item is to be removed.

SubMenuEventID

[in] Specifies the client defined sub-menu event ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

If a sub-menu item is deleted from the middle of the sub-menu item list, the list will contract.

See Also

- [SimConnect_MenuAddItem](#)
 - [SimConnect_MenuAddSubItem](#)
 - [SimConnect_MenuDeleteItem](#)
 - [SimConnect API Reference](#)
-

SimConnect_Text

The **SimConnect_Text** function is used to display a text menu, message window, or scrolling or static text, on the screen.

Syntax

```
HRESULT SimConnect_Text(  
    HANDLE hSimConnect,  
    SIMCONNECT_TEXT_TYPE type,  
    float fTimeSeconds,  
    SIMCONNECT_CLIENT_EVENT_ID EventID,  
    DWORD cbUnitSize,  
    void* pDataSet  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

type

[in] One member of the [SIMCONNECT_TEXT_TYPE](#) enumeration type.

fTimeSeconds

[in] The timeout value for the text, message, or menu in seconds. If zero is entered, the text, message, or menu will not timeout. For text and messages, this timeout value can be overridden if there are other text requests waiting in the queue (see the **Remarks** below). If the timeout requested exceeds the minimum display time, and another text request is waiting, the timeout value is overridden and the text will only be displayed for the minimum display time. The default minimum display time is two seconds for static text and 10 seconds for scrolling text, and 5 seconds for message windows. These can be changed in the [SimConnect] section of Prepar3D.cfg file using the **TextMinPrintTimeSeconds**, **TextMinScrollTimeSeconds**, and **TextMinMessageWindowTimeSeconds** settings.

EventID

[in] Specifies the client defined event ID, which will be returned along with the [SIMCONNECT_TEXT_RESULT](#) (in the dwData parameter) of a [SIMCONNECT_RECV_EVENT](#) structure.

cbUnitSize

[in] Specifies the size of pDataSet in bytes.

pDataSet

[in] Specifies the array of string data for the menu or text. For text and message windows simply enter the string. For a menu, the format of the string is a list of null-terminated string entries, with the menu title and prompt as the first two entries, for example: "Title\0Prompt\0Menu item 1\0Menu item 2\0", with a maximum of ten menu items. If an empty string is sent in pDataSet along with an EventID that matches a menu or text in the queue (see **Remarks** below), that entry will be removed from the queue, with the **SIMCONNECT_TEXT_RESULT_REMOVED** event being returned to the client. If a new set of menu items, or new text string, is sent with an EventID that matches an entry in the queue, that entry will be replaced in the queue, with the **SIMCONNECT_TEXT_RESULT_REPLACE** event being returned to the client. The entry will not lose its place in the queue. This change will take place even if the text or menu is being rendered on the screen.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary samples [Text Menu](#)
[FacilitiesData](#)

Remarks

Message windows, text, and menus can all be displayed simultaneously, however only one instance of each is allowed at any given time. Requests are queued as they are received, with only the topmost in the queue being displayed. There is one queue for menus and another for static and scrolling text.

When a request is first displayed the **SIMCONNECT_TEXT_RESULT_DISPLAYED** event will be sent to the client. If a request joins the queue and cannot immediately be displayed the event

SIMCONNECT_TEXT_RESULT_QUEUED will be sent. When it is the turn of the new request to be displayed the **SIMCONNECT_TEXT_RESULT_DISPLAYED** event will be sent. If the request is for a menu, and the user selects one of the menu entries, one of the

SIMCONNECT_TEXT_RESULT_MENU_SELECT_N events will be returned (see the [SIMCONNECT_TEXT_TYPE](#) enumeration), and the menu closed.

The default location for message windows and static or scrolling text is along the top of the screen. A user can move and resize the window that the text is being displayed in, but it is not possible to specify an alternative location for the text programmatically.

When a message is sent, a **TextEventCreated** event is fired and can be listened for with [SimConnect_SubscribeToSystemEvent](#)

See Also

- [SimConnect_MenuAddItem](#)
- [SimConnect API Reference](#)

[- top -](#)

SimObject Functions

Overview

To view a list of all SimObject SimConnect functions, see the [SimObject Functions](#) table.

General SimObject Functions

SimConnect_SetDataOnSimObject

The **SimConnect_SetDataOnSimObject** function is used to make changes to the data properties of an object.

Syntax

```
HRESULT SimConnect_SetDataOnSimObject(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_DEFINITION_ID DefineID,  
    SIMCONNECT_OBJECT_ID ObjectID,  
    SIMCONNECT_DATA_SET_FLAG Flags,  
    DWORD ArrayCount,  
    DWORD cbUnitSize,  
    void* pDataSet  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

DefineID

[in] Specifies the ID of the client defined data definition.

ObjectID

[in] Specifies the ID of the *Prepar3D* object that the data should be about. This ID can be **SIMCONNECT_OBJECT_ID_USER** (to specify the user's aircraft) or obtained from a **SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE** structure after a call to **SimConnect_RequestDataOnSimObjectType**. Also refer to the [note on multiplayer mode](#) in the remarks for **SimConnect_RequestDataOnSimObject**.

Flags

[in] Null, or one or more of the following flags.

Flag	Description
SIMCONNECT_DATA_SET_FLAG_TAGGED	The data to be set is being sent in tagged format. Refer to SimConnect_RequestDataOnSimObject for more details on the tagged format.
SIMCONNECT_DATA_SET_FLAG_UNBLOCK	Any currently outstanding blocking callbacks are released. See SimConnect_RequestSynchronousBlock for more information.

ArrayCount

[in] Specifies the number of elements in the data array. A count of zero is interpreted as one element. Ensure that the data array has been initialized completely before transmitting it to *Prepar3D*. Failure to properly initialize all array elements may result in unexpected behavior.

cbUnitSize

[in] Specifies the size of each element in the data array in bytes. The data alignment of the individual element structures must align with the data layout defined by calls to the **SimConnect>AddToDataDefinition** function for the given definition (*DefineID*). Note that the default **SIMCONNECT_DATATYPE** value for the **SimConnect>AddToDataDefinition** *DatumType* parameter is **SIMCONNECT_DATATYPE_FLOAT64** and must be correctly set if your element structures contain other data type members (i.e. **int** or **float**).

pDataSet

[in] Pointer to the data that is to be written. If the data is not in tagged format, this should point to the block of data. If the data is in tagged format this should point to the first tag name (*datumID*), which is always four bytes long, which should be followed by the data itself. Any number of tag name/value pairs can be specified this way, the server will use the *cbUnitSize* parameter to determine how much data has been sent.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
static enum DATA_DEFINE_ID {
    DEFINITION3
};

// Link DEFINITION3 with the SIMCONNECT_DATA_INITPOSITION structure

hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION3, "Initial Position", "NULL",
    SIMCONNECT_DATATYPE_INITPOSITION, 0);

SIMCONNECT_DATA_INITPOSITION Init;
Init.Altitude = 5000.0;
Init.Latitude = 47.64210;
Init.Longitude = -122.13010;
Init.Pitch = -0.0;
Init.Bank = -1.0;
Init.Heading = 180.0;
Init.OnGround = 0;
Init.Airspeed = 0;

SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION3,
    SIMCONNECT_OBJECT_ID_USER, 0, 0, sizeof(Init), &Init);
```

Working Samples

Primary samples [Set Data](#)
[Throttle Control](#)

Reference samples [AI Objects and Waypoints](#)
[Managed AI Waypoints](#)

Remarks

The data that is set on an object is defined in a data definition (see the [SimConnect_AddToDataDefinition](#) function). This data can include the following structures: **SIMCONNECT_DATA_WAYPOINT**, **SIMCONNECT_DATA_INITPOSITION**, and **SIMCONNECT_DATA_MARKERSTATE**. Any number of waypoints can be given to an AI object using a single call to this function, and any number of marker state structures can also be combined into an array.

The [Simulation Variables](#) document includes a column indicating whether variables can be written to or not. An exception will be returned if an attempt is made to write to a variable that cannot be set in this way.

See Also

- [SimConnect_AddToDataDefinition](#)
- [SimConnect_ClearDataDefinition](#)

-
- [SimConnect_RequestDataOnSimObject](#)
 - [SimConnect_RequestDataOnSimObjectType](#)
 - [SimConnect API Reference](#)
-

SimConnect_ChangeVehicle

The **SimConnect_ChangeVehicle** function is used to change the selected vehicle.

Syntax

```
HRESULT SimConnect_ChangeVehicle(  
    HANDLE hSimConnect,  
    const char* Vehicletitle  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
Vehicletitle
[in] title of the desired vehicle

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Change Vehicle](#)
[Managed Change Vehicle](#)

Remarks

The instant replay does not get reset on the vehicle change which is the normal behavior when changing the vehicle in Prepar3D.

See Also

None.

SimConnect_RequestAttachPointData

The **SimConnect_RequestAttachPointData** function is used to request attach point data from a simulation object.

Syntax

```
HRESULT SimConnect_RequestAttachPointData(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    DWORD dwObjectID,  
    const char* szAttachPointName,  
    BOOL bRequestWorldCoordinates = FALSE  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client-defined request. This is used later by the client to identify which data has been received. This value should be unique for each request, re-using a *RequestID* will overwrite any previous request using the same ID.

dwObjectID

[in] The ID of the object to request attach point information from. This ID can be

SIMCONNECT_OBJECT_ID_USER (to specify the user's aircraft) or obtained from a **SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE** structure after a call to [SimConnect_RequestDataOnSimObjectType](#).

szAttachPointName

[in] Specifies the name of the attach point to request data from.

bRequestWorldCoordinates

[in, optional] If set to TRUE, the world position and orientation information will be requested as well. If FALSE, world position and orientation information will be returned as zero.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The data will be returned in a [SIMCONNECT_RECV_ATTACHPOINT_DATA](#) structure.

See Also

- [SimConnect API Reference](#)

SimConnect_RequestDataOnSimObject

The **SimConnect_RequestDataOnSimObject** function is used to request when the SimConnect client is to receive data values for a specific object.

Syntax

```
HRESULT SimConnect_RequestDataOnSimObject(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    SIMCONNECT_DATA_DEFINITION_ID DefineID,  
    SIMCONNECT_OBJECT_ID ObjectID,  
    SIMCONNECT_PERIOD Period,  
    SIMCONNECT_DATA_REQUEST_FLAG Flags = 0,  
    DWORD origin = 0,  
    DWORD interval = 0,  
    DWORD limit = 0  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which data has been received. This value should be unique for each request, re-using a *RequestID* will overwrite any previous request using the same ID.

DefineID

	[in] Specifies the ID of the client defined data definition.										
ObjectID	<p>[in] Specifies the ID of the <i>Prepar3D</i> object that the data should be about. This ID can be SIMCONNECT_OBJECT_ID_USER (to specify the user's aircraft) or obtained from a SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE structure after a call to SimConnect_RequestDataOnSimObjectType. Also refer to the note on developing clients for multiplayer mode in the Remarks section below.</p>										
Period	[in] One member of the SIMCONNECT_PERIOD enumeration type, specifying how often the data is to be sent by the server and received by the client.										
Flags	<p>[in, optional] A DWORD containing one or more of the following values:</p> <table border="0"> <thead> <tr> <th style="text-align: center;">Flag value</th> <th style="text-align: center;">Description</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td>The default, data will be sent strictly according to the defined period.</td> </tr> <tr> <td style="text-align: center;">SIMCONNECT_DATA_REQUEST_FLAG_CHANGED</td> <td>Data will only be sent to the client when one or more values have changed. If this is the only flag set, then all the variables in a data definition will be returned if just one of the values changes.</td> </tr> <tr> <td style="text-align: center;">SIMCONNECT_DATA_REQUEST_FLAG_TAGGED</td> <td>The following types support the SIMCONNECT_DATA_REQUEST_FLAG_CHANGED: <ul style="list-style-type: none"> • SIMCONNECT_DATATYPE_INT32 • SIMCONNECT_DATATYPE_FLOAT32 • SIMCONNECT_DATATYPE_INT64 • SIMCONNECT_DATATYPE_FLOAT64 • SIMCONNECT_DATATYPE_XYZ • SIMCONNECT_DATATYPE_PBH • SIMCONNECT_DATATYPE_LATLONALT • SIMCONNECT_DATATYPE_STRING8 • SIMCONNECT_DATATYPE_STRING32 • SIMCONNECT_DATATYPE_STRING64 • SIMCONNECT_DATATYPE_STRING128 • SIMCONNECT_DATATYPE_STRING256 • SIMCONNECT_DATATYPE_STRING260 </td> </tr> <tr> <td style="text-align: center;">SIMCONNECT_DATA_REQUEST_FLAG_BLOCK</td> <td>Requested data will be sent in tagged format (datum ID/value pairs). Tagged format requires that a datum reference ID is returned along with the data value, in order that the client code is able to identify the variable. This flag is usually set in conjunction with the previous flag, but it can be used on its own to return all the values in a data definition in datum ID/value pairs. See the SIMCONNECT_RECV_SIMOBJECT_DATA structure for more details.</td> </tr> </tbody> </table>	Flag value	Description	0	The default, data will be sent strictly according to the defined period.	SIMCONNECT_DATA_REQUEST_FLAG_CHANGED	Data will only be sent to the client when one or more values have changed. If this is the only flag set, then all the variables in a data definition will be returned if just one of the values changes.	SIMCONNECT_DATA_REQUEST_FLAG_TAGGED	The following types support the SIMCONNECT_DATA_REQUEST_FLAG_CHANGED : <ul style="list-style-type: none"> • SIMCONNECT_DATATYPE_INT32 • SIMCONNECT_DATATYPE_FLOAT32 • SIMCONNECT_DATATYPE_INT64 • SIMCONNECT_DATATYPE_FLOAT64 • SIMCONNECT_DATATYPE_XYZ • SIMCONNECT_DATATYPE_PBH • SIMCONNECT_DATATYPE_LATLONALT • SIMCONNECT_DATATYPE_STRING8 • SIMCONNECT_DATATYPE_STRING32 • SIMCONNECT_DATATYPE_STRING64 • SIMCONNECT_DATATYPE_STRING128 • SIMCONNECT_DATATYPE_STRING256 • SIMCONNECT_DATATYPE_STRING260 	SIMCONNECT_DATA_REQUEST_FLAG_BLOCK	Requested data will be sent in tagged format (datum ID/value pairs). Tagged format requires that a datum reference ID is returned along with the data value, in order that the client code is able to identify the variable. This flag is usually set in conjunction with the previous flag, but it can be used on its own to return all the values in a data definition in datum ID/value pairs. See the SIMCONNECT_RECV_SIMOBJECT_DATA structure for more details.
Flag value	Description										
0	The default, data will be sent strictly according to the defined period.										
SIMCONNECT_DATA_REQUEST_FLAG_CHANGED	Data will only be sent to the client when one or more values have changed. If this is the only flag set, then all the variables in a data definition will be returned if just one of the values changes.										
SIMCONNECT_DATA_REQUEST_FLAG_TAGGED	The following types support the SIMCONNECT_DATA_REQUEST_FLAG_CHANGED : <ul style="list-style-type: none"> • SIMCONNECT_DATATYPE_INT32 • SIMCONNECT_DATATYPE_FLOAT32 • SIMCONNECT_DATATYPE_INT64 • SIMCONNECT_DATATYPE_FLOAT64 • SIMCONNECT_DATATYPE_XYZ • SIMCONNECT_DATATYPE_PBH • SIMCONNECT_DATATYPE_LATLONALT • SIMCONNECT_DATATYPE_STRING8 • SIMCONNECT_DATATYPE_STRING32 • SIMCONNECT_DATATYPE_STRING64 • SIMCONNECT_DATATYPE_STRING128 • SIMCONNECT_DATATYPE_STRING256 • SIMCONNECT_DATATYPE_STRING260 										
SIMCONNECT_DATA_REQUEST_FLAG_BLOCK	Requested data will be sent in tagged format (datum ID/value pairs). Tagged format requires that a datum reference ID is returned along with the data value, in order that the client code is able to identify the variable. This flag is usually set in conjunction with the previous flag, but it can be used on its own to return all the values in a data definition in datum ID/value pairs. See the SIMCONNECT_RECV_SIMOBJECT_DATA structure for more details.										
origin	[in, optional] The number of <i>Period</i> events that should elapse before transmission of the data begins. The default is zero, which means transmissions will start immediately.										
interval	[in, optional] The number of <i>Period</i> events that should elapse between transmissions of the data. The default is zero, which means the data is transmitted every <i>Period</i> .										
limit	[in, optional] The number of times the data should be transmitted before this communication is ended. The default is zero, which means the data should be transmitted endlessly.										
Return Values											
The function returns an HRESULT . Possible values include, but are not limited to, those in the following table.											
<table border="0"> <thead> <tr> <th style="text-align: left;">Return value</th> <th style="text-align: left;">Description</th> </tr> </thead> </table>		Return value	Description								
Return value	Description										

S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```

static enum DATA_DEFINE_ID {
    DEFINITION_1,
    DEFINITION_2
};

static enum DATA_REQUEST_ID {
    REQUEST_1,
    REQUEST_2,
};

struct Struct1
{
    double kohlsmann;
    double altitude;
    double latitude;
    double longitude;
};

// Match string definitions from the Simulation Variables document with the client defined ID

hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Kohlsman setting hg",
"inHg");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Indicated Altitude", "feet");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Latitude", "degrees");
hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Longitude", "degrees");

// Sections of code in DispatchProc

.....
SimConnect_RequestDataOnSimObject(hSimConnect, REQUEST_2, DEFINITION_1,
SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD_SECOND);
.....
// When the data is received -- cast it to the correct structure type

case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:
{
    SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData =
(SIMCONNECT_RECV_SIMOBJECT_DATA*) pData;

    switch(pObjData->dwRequestID)
    {
        case REQUEST_2:

            Struct1 *pS = (Struct1*)&pObjData->dwData;

            // Add code to process the structure appropriately

            break;
    }
    break;
}
.....

```

Working Samples

Primary samples [Tagged Data](#)
Reference samples [Weather Station](#)

Remarks

Changing the *Period* parameter or changing the content of a data definition has a higher performance cost than changing the *origin*, *interval* or *limit* parameters. So to temporarily turn off data requests, especially for short periods of time, consider setting the *interval* parameter to a very high value (such as **DWORD _MAX**). If changes are required to a data definition, consider setting the *Period* parameter to **SIMCONNECT_PERIOD_NEVER** (see the [SIMCONNECT_PERIOD](#) enumeration) before making the changes, and then turning on the appropriate period after the changes have been made.

It is possible to change the period of a request, by resending the **SimConnect_RequestDataOnSimObject** call with the same *RequestID*, *DefineID* and *ObjectID* parameters, but with a new period. The one exception to this is the new period cannot be **SIMCONNECT_PERIOD_ONCE**, in this case a request with a new *RequestID* should be sent.

If an object is removed, and if it has a current periodic data request, a **SIMCONNECT_EXCEPTION_UNRECOGNIZED_ID** exception will be sent every requested period. This can be avoided by sending a new [SimConnect_RequestDataOnSimObject](#) with the period set to **SIMCONNECT_PERIOD_NEVER**, before the object is removed or once the first exception is generated.

Data is always transmitted with the [SimConnect_RequestDataOnSimObject](#) function, so if timing only notifications are required, use the [SimConnect_SubscribeToSystemEvent](#) function.

Note that variable length strings should not be used in data definitions, except where the *Period* parameter has been set to **SIMCONNECT_PERIOD_ONCE**.

One method of testing whether the user has changed aircraft type is to use this function to return the title of the user aircraft, and note that if it changes, the user has changed the type of aircraft (all aircraft types have unique title strings, including those simply with different color schemes). An example of requesting the aircraft title is in the [Variable Strings](#) working sample. See the [Aircraft Configuration Files](#) document for more details on titles.

If boolean data has been requested as part of a data definition, note that the only reliable numeric equivalent is that 0 is returned for False. Non-zero values, especially both 1 and -1, are used to indicate True.

Note: Multiplayer Mode

When developing a client for use in multiplayer mode it is not safe to use the ID number for the user aircraft returned by the function [SimConnect_RequestDataOnSimObjectType](#), as the actual number can change depending on several factors, including the number of users involved in the multiplayer scenario. Always use the constant **SIMCONNECT_OBJECT_ID_USER** for the *ObjectID* parameter if the SimConnect client is to work in multiplayer mode. Also note that in this mode it is good practice to remove any calls associated with periodic data on AI objects and to remove subscriptions to AI objects.

See Also

- [SimConnect API Reference](#)
- [SimConnect_AddToDataDefinition](#)
- [SimConnect_ClearDataDefinition](#)
- [SimConnect_RequestDataOnSimObjectType](#)

SimConnect_RequestDataOnSimObjectType

The **SimConnect_RequestDataOnSimObjectType** function is used to retrieve information about simulation objects of a given type that are within a specified radius of the user's aircraft.

Syntax

```
HRESULT SimConnect_RequestDataOnSimObjectType(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    SIMCONNECT_DATA_DEFINITION_ID DefineID,  
    DWORD dwRadiusMeters,
```

```
SIMCONNECT_SIMOBJECT_TYPE type  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which data has been received. This value should be unique for each request, re-using a RequestID will overwrite any previous request using the same ID.

DefineID

[in] Specifies the ID of the client defined data definition.

dwRadiusMeters

[in] Double word containing the radius in meters. If this is set to zero only information on the user aircraft will be returned, although this value is ignored if type is set to

SIMCONNECT_SIMOBJECT_TYPE_USER. The error

SIMCONNECT_EXCEPTION_OUT_OF_BOUNDS will be returned if a radius is given and it exceeds the maximum allowed (200000 meters, or 200 Km).

type

[in] Specifies the type of object to receive information on. One member of the **SIMCONNECT_SIMOBJECT_TYPE** enumeration type.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

See [SimConnect_AddToDataDefinition](#) for an example of when to use this function

Working Samples

Primary samples [Request Data](#)
[Variable Strings](#)

Reference samples

Remarks

The data will be returned on all the relevant objects within the specified radius, but they will not be returned in any specific order. It is the responsibility of the client program to sort the returned data into order, if that is required. Information is returned in a

[SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE](#) structure, one structure per object.

Also refer to the [note on multiplayer mode](#) in the remarks for [SimConnect_RequestDataOnSimObject](#).

See Also

- [SimConnect API Reference](#)
- [SimConnect_AddToDataDefinition](#)
- [SimConnect_ClearDataDefinition](#)
- [SimConnect_RequestDataOnSimObject](#)

AI Object Specific Functions

[SimConnect_AICreateEnrouteATCAircraft](#)

The **SimConnect_AICreateEnrouteATCAircraft** function is used to create an AI controlled aircraft that is about to start or is already underway on its flight plan.

Syntax

```
HRESULT SimConnect_AICreateEnrouteATCAircraft(  
    HANDLE hSimConnect,  
    const char* szContainertitle,  
    const char* szTailNumber,  
    int iFlightNumber,  
    const char* szFlightPlanPath,  
    double dFlightPlanPosition,  
    BOOL bTouchAndGo,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szContainertitle

[in] Null-terminated string containing the container title. The container title is found in the aircraft.cfg file (see the [Aircraft Configuration Files](#) document): for example: **title=Mooney Bravo** or **title=Commercial Airliner**.

szTailNumber

[in] Null-terminated string containing the tail number. This should have a maximum of 12 characters.

iFlightNumber

[in] Integer containing the flight number. There is no specific maximum length of this number. Any negative number indicates that there is no flight number.

szFlightPlanPath

[in] Null-terminated string containing the path to the flight plan file. Flight plans have the extension .pln, but no need to enter an extension here. The easiest way to create flight plans is to create them from within *Prepar3D* itself, and then save them off for use with the AI controlled aircraft.

There is no need to enter the full path to the file (just enter the filename) if the flight plan is in the default *Prepar3D v5 Files* directory.

dFlightPlanPosition

[in] Double floating point number containing the flight plan position. The number before the point contains the waypoint index, and the number afterwards how far along the route to the next waypoint the aircraft is to be positioned. The first waypoint index is 0. For example, **0.0** indicates that the aircraft has not started on the flight plan, **2.5** would indicate the aircraft is to be initialized halfway between the third and fourth waypoints (which would have indexes 2 and 3). The waypoints are those recorded in the flight plan, which may just be two airports, and do not include any taxiway points on the ground. Also there is a threshold that will ignore requests to have an aircraft taxiing or taking off, or landing. So set the value after the point to ensure the aircraft will be in level flight. See the section on [Aircraft Flight Plans](#).

bTouchAndGo

[in] Flag, **True** indicating that landings should be touch and go, and not full stop landings.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

[Primary sample](#) [AI Traffic](#)

Remarks

An enroute aircraft can be on the ground or airborne when it is created by this function. Typically this will be an aircraft flying under IFR rules, and in constant radio contact with ATC. A number of errors, including **SIMCONNECT_EXCEPTION_CREATE_AIRCRAFT_FAILED**, apply to AI objects (refer to the [SIMCONNECT_EXCEPTION](#) enum for more details).

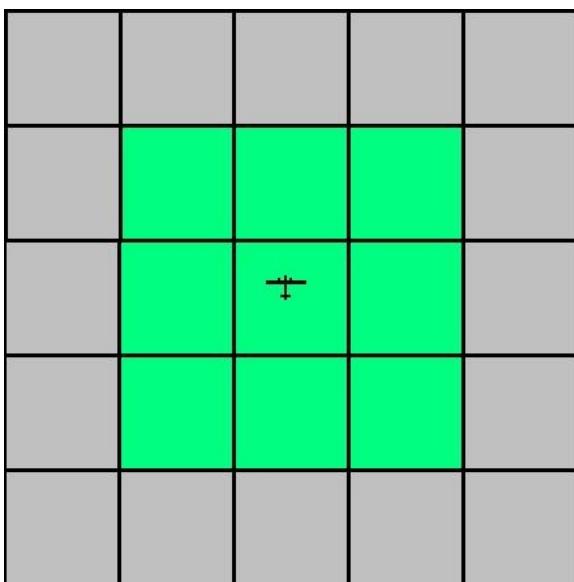
A **SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE** event notification can be subscribed to (see the [SimConnect_SubscribeToSystemEvent](#) function), which will return a **SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE** structure whenever any client, including the one making the change, successfully adds or removes an AI controlled object.

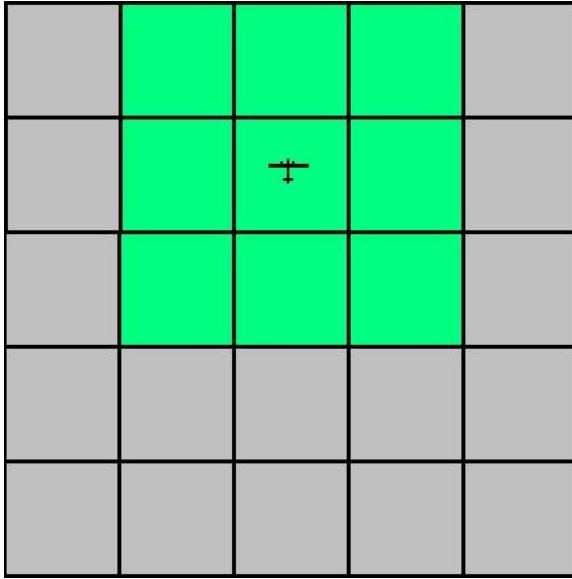
This function should be used for fixed-wing aircraft flying between airports on land. There is no internal AI pilot for helicopters, gliders or hot-air balloons. In order to add a helicopter, glider or balloon not controlled by the user, the SimConnect client must implement full control of the aircraft. Set up these objects with a call to [SimConnect_AICreateSimulatedObject](#).

For float-planes the recommended procedure is to control them using waypoints, and not the ATC system, as there is no concept of a "parking space" after a water landing. So, the waypoints of the route of the float-plane should include the route that it should follow before takeoff and after landing. For all these cases of controlling aircraft using the client, or using waypoints, set up the object using the [SimConnect_AICreateNonATCAircraft](#) call.

The Reality Bubble

In order to improve performance, only a certain area around the user's aircraft is actually simulated at any one time. As the user flies the aircraft that area (referred to in the simulator as the "reality bubble") moves along with the aircraft. Simulated objects outside of the area are removed altogether, and new AI aircraft are not created if they fall outside these bounds. The reality bubble is in fact more of a box, and is aligned to lines of latitude and longitude, so the box becomes narrower near the poles than it is as the equator. The following diagram shows how the reality bubble works. There are always nine boxes (approximately 64Km square at the equator) that are simulated, and the user aircraft is always in the center box. As the aircraft flies over a boundary line, then the areas that are simulated change. In the following diagram, the green areas are simulated (active aircraft, airports, ground vehicles, shipping traffic and so on), and the grey areas are not.





Note that if an aircraft, or other simulation object, falls outside this area and is deleted by the system, the client will no longer receive information on the object (typically requested by the [SimConnect_RequestDataOnSimObject](#) function). Consider using the [SimConnect_SubscribeToSystemEvent](#) function to request notifications for "ObjectRemoved" to test for these situations. No automatic notification is given when information is requested on an object, and then subsequently that object is removed.

Aircraft Flight Plans

The following table contains an example flight plan from Sea-Tac International airport (KSEA) to San Francisco International airport (KSFO), following high-altitude airways. There are six waypoints, including the departure and destination airports. A flight plan can be created by *Prepar3D* or created directly in XML, matching the required format. The simplest flight plan will contain only the departure and destination airports. The **ATCWaypoints** in a scenario file are not identical to the **SIMCONNECT_DATA_WAYPOINT** structures. **ATCWaypoints** in a flightplan can be very far apart, and the AI pilot of *Prepar3D* requires waypoints to be no more than around five miles apart, so a flightplan such as the one given below will be broken down by the AI component into shorter sections -- using the **SIMCONNECT_DATA_WAYPOINT** structures.

```
<?xml version="1.0" encoding="UTF-8"?>

<SimBase.Document Type="AceXML" version="1,0">
  <Descr>AceXML Document</Descr>
  <WorldBase.FlightPlan>
    <title>KSEA to KSFO</title>
    <FPType>IFR</FPType>
    <RouteType>HighAlt</RouteType>
    <CruisingAlt>31000</CruisingAlt>
    <DepartureID>KSEA</DepartureID>
    <DepartureLLA>N47 25' 53.27",W122 18' 28.83",+000433.00</DepartureLLA>
    <DestinationID>KSFO</DestinationID>
    <DestinationLLA>N37 36' 26.00",W122 22' 49.59",+000013.00</DestinationLLA>
    <Descr>KSEA, KSFO</Descr>
    <DeparturePosition>34R</DeparturePosition>
    <DepartureName>Seattle-Tacoma Intl</DepartureName>
    <DestinationName>San Francisco Intl</DestinationName>
    <AppVersion>
      <AppVersionMajor>10</AppVersionMajor>
      <AppVersionBuild>60327</AppVersionBuild>
    </AppVersion>

    <ATCWaypoint id="KSEA">
      <ATCWaypointType>Airport</ATCWaypointType>
      <WorldPosition>N47 25' 53.27",W122 18' 28.83",+000433.00</WorldPosition>
      <ICAO>
        <ICAOIdent>KSEA</ICAOIdent>
```

```

        </ICAO>
    </ATCWaypoint>

    <ATCWaypoint id="SEA">
        <ATCWaypointType>VOR</ATCWaypointType>
        <WorldPosition>N47 26' 7.36",W122 18' 34.59",+000000.00</WorldPosition>
        <ICAO>
            <ICAORegion>K1</ICAORegion>
            <ICAOIdent>SEA</ICAOIdent>
        </ICAO>
    </ATCWaypoint>

    <ATCWaypoint id="LMT">
        <ATCWaypointType>VOR</ATCWaypointType>
        <WorldPosition>N42 9' 11.34",W121 43' 39.10",+000000.00</WorldPosition>
        <ATCAirway>J65</ATCAirway>
        <ICAO>
            <ICAORegion>K1</ICAORegion>
            <ICAOIdent>LMT</ICAOIdent>
        </ICAO>
    </ATCWaypoint>

    <ATCWaypoint id="RBL">
        <ATCWaypointType>VOR</ATCWaypointType>
        <WorldPosition>N40 5' 56.07",W122 14' 10.88",+000000.00</WorldPosition>
        <ATCAirway>J65</ATCAirway>
        <ICAO>
            <ICAORegion>K2</ICAORegion>
            <ICAOIdent>RBL</ICAOIdent>
        </ICAO>
    </ATCWaypoint>

    <ATCWaypoint id="OAK">
        <ATCWaypointType>VOR</ATCWaypointType>
        <WorldPosition>N37 43' 33.30",W122 13' 24.92",+000000.00</WorldPosition>
        <ATCAirway>J3</ATCAirway>
        <ICAO>
            <ICAORegion>K2</ICAORegion>
            <ICAOIdent>OAK</ICAOIdent>
        </ICAO>
    </ATCWaypoint>

    <ATCWaypoint id="KSFO">
        <ATCWaypointType>Airport</ATCWaypointType>
        <WorldPosition>N37 36' 26.00",W122 22' 49.59",+000013.00</WorldPosition>
        <ICAO>
            <ICAOIdent>KSFO</ICAOIdent>
        </ICAO>
    </ATCWaypoint>

</WorldBase.FlightPlan>
</SimBase.Document>

```

See Also

- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateParkedATCAircraft](#)
- [SimConnect_AICreateSimulatedObject](#)
- [SimConnect_AIRemoveObject](#)
- [SimConnect_AISetAircraftFlightPlan](#)
- [SimConnect API Reference](#)

SimConnect_AICreateEnrouteATCAircraftW

The **SimConnect_AICreateEnrouteATCAircraftW** function is used to create an AI controlled aircraft that is about to start or is already underway on its flight plan. This version of the functions supports

unicode file paths, titles, tail numbers, and flight plans.

Syntax

```
HRESULT SimConnect_AICreateEnrouteATCAircraftW(
    HANDLE hSimConnect,
    const wchar_t* szContainertitle,
    const wchar_t* szTailNumber,
    int iFlightNumber,
    const wchar_t* szFlightPlanPath,
    double dFlightPlanPosition,
    BOOL bTouchAndGo,
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szContainertitle

[in] Null-terminated string containing the container title. The container title is found in the aircraft.cfg file (see the [Aircraft Configuration Files](#) document): for example: **title=Mooney Bravo** or **title=Commercial Airliner**.

szTailNumber

[in] Null-terminated string containing the tail number. This should have a maximum of 12 characters.

iFlightNumber

[in] Integer containing the flight number. There is no specific maximum length of this number. Any negative number indicates that there is no flight number.

szFlightPlanPath

[in] Null-terminated string containing the path to the flight plan file. Flight plans have the extension .pln, but no need to enter an extension here. The easiest way to create flight plans is to create them from within *Prepar3D* itself, and then save them off for use with the AI controlled aircraft. There is no need to enter the full path to the file (just enter the filename) if the flight plan is in the default *Prepar3D v5 Files* directory.

dFlightPlanPosition

[in] Double floating point number containing the flight plan position. The number before the point contains the waypoint index, and the number afterwards how far along the route to the next waypoint the aircraft is to be positioned. The first waypoint index is 0. For example, **0.0** indicates that the aircraft has not started on the flight plan, **2.5** would indicate the aircraft is to be initialized halfway between the third and fourth waypoints (which would have indexes 2 and 3). The waypoints are those recorded in the flight plan, which may just be two airports, and do not include any taxiway points on the ground. Also there is a threshold that will ignore requests to have an aircraft taxiing or taking off, or landing. So set the value after the point to ensure the aircraft will be in level flight. See the section on [Aircraft Flight Plans](#).

bTouchAndGo

[in] Flag, **True** indicating that landings should be touch and go, and not full stop landings.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value Description

S_OK The function succeeded.

E_FAIL The function failed.

See Also

- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateParkedATCAircraft](#)
- [SimConnect_AICreateSimulatedObject](#)

-
- [SimConnect_AIRemoveObject](#)
 - [SimConnect_AISetAircraftFlightPlan](#)
 - [SimConnect API Reference](#)
-

SimConnect_AICreateNonATCAircraft

The **SimConnect_AICreateNonATCAircraft** function is used to create an aircraft that is not flying under ATC control (so is typically flying under VFR rules).

Syntax

```
HRESULT SimConnect_AICreateNonATCAircraft(  
    HANDLE hSimConnect,  
    const char* szContainertitle,  
    const char* szTailNumber,  
    SIMCONNECT_DATA_INITPOSITION InitPos,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szContainertitle

[in] Null-terminated string containing the container title. The container title is case-sensitive and can be found in the aircraft.cfg file (see the [Aircraft Configuration Files](#) document): for example:
title=Mooney Bravo or **title=Commercial Airliner**.

szTailNumber

[in] Null-terminated string containing the tail number. This should have a maximum of 12 characters.

InitPos

[in] Specifies the initial position, using a [SIMCONNECT_DATA_INITPOSITION](#) structure.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [AI Objects and Waypoints](#)
[Managed AI Waypoints](#)

Remarks

A non-ATC aircraft can be on the ground or airborne when it is created by this function. A number of errors, including **SIMCONNECT_EXCEPTION_CREATE_AIRCRAFT_FAILED**, apply to AI objects (refer to the [SIMCONNECT_EXCEPTION](#) enum for more details).

Refer to the remarks for [SimConnect_AICreateEnrouteATCAircraft](#).

See Also

- [SimConnect_AICreateEnrouteATCAircraft](#)
 - [SimConnect_AICreateParkedATCAircraft](#)
 - [SimConnect_AICreateSimulatedObject](#)
-

-
- [SimConnect_AIRemoveObject](#)
 - [SimConnect_AISetAircraftFlightPlan](#)
 - [SimConnect API Reference](#)
-

SimConnect_AICreateParkedATCAircraft

The **SimConnect_AICreateParkedATCAircraft** function is used to create an AI controlled aircraft that is currently parked and does not have a flight plan.

Syntax

```
HRESULT SimConnect_AICreateParkedATCAircraft(  
    HANDLE hSimConnect,  
    const char* szContainertitle,  
    const char* szTailNumber,  
    const char* szAirportID,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szContainertitle

[in] Null-terminated string containing the container title. The container title is case-sensitive and can be found in the aircraft.cfg file (see the [Aircraft Configuration Files](#) document): for example:
title=Mooney Bravo or **title=Commercial Airliner**.

szTailNumber

[in] Null-terminated string containing the tail number. This should have a maximum of 12 characters.

szAirportID

[in] Null-terminated string containing the airport ID. This is the ICAO identification string, for example, KSEA for SeaTac International.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [AI Traffic](#)

Remarks

Calling this function is no guarantee that there is sufficient parking space at the specified airport. An error will be returned if there is insufficient parking space, and an aircraft will not be created. A number of errors, including **SIMCONNECT_EXCEPTION_CREATE_AIRCRAFT_FAILED**, apply to AI objects (refer to the [SIMCONNECT_EXCEPTION](#) enum for more details). After creating an aircraft with this function, a call to [SimConnect_AISetAircraftFlightPlan](#) will set the aircraft in motion.

When allocating a parking space to an aircraft, the simulation uses a radius based on half the wingspan defined in the [aircraft_geometry] section of the [Aircraft Configuration File](#). Use the Dump Airport List tool (refer to the [Traffic Toolbox](#) documentation) to see what parking spaces are available at any airport.

Refer also to the remarks for [SimConnect_AICreateEnrouteATCAircraft](#).

See Also

- [SimConnect_AICreateEnrouteATCAircraft](#)
- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateSimulatedObject](#)
- [SimConnect_AIRemoveObject](#)
- [SimConnect_AISetAircraftFlightPlan](#)
- [SimConnect API Reference](#)

SimConnect_AICreateSimulatedObject

The **SimConnect_AICreateSimulatedObject** function is used to create AI controlled objects other than aircraft.

Syntax

```
HRESULT SimConnect_AICreateSimulatedObject(  
    HANDLE hSimConnect,  
    const char* szContainertitle,  
    SIMCONNECT_DATA_INITPOSITION InitPos,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szContainertitle

[in] Null-terminated string containing the container title. The container title is case-sensitive and can be found in the sim.cfg file, for example:

Object Types	Examples
Ground Vehicles	title=Automobile
	title=FuelTruck
Boats	title=VEH_air_bagcart1
Miscellaneous	title=VEH_air_bagcart_FlatNosed
	title=cargoA
	title=cargoA_hoop
	title=cargoB
	title=Flour_bomb_floating_dock
	title=HumpbackWhale

InitPos

[in] Specifies the initial position, using a [SIMCONNECT_DATA_INITPOSITION](#) structure.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [AI Objects and Waypoints](#)
[Managed AI Waypoints](#)

Remarks

This function can be used to create a stationary aircraft (such as an unflyable aircraft on display outside a flight museum), but is typically intended to create simulation objects other than aircraft (such as ground vehicles, boats, and a number of special objects such as humpback whales and hot-air balloons). A number of errors apply to AI objects (refer to the [SIMCONNECT_EXCEPTION](#) enum for more details).

See Also

- [SimConnect_AICreateEnrouteATCAircraft](#)
 - [SimConnect_AICreateNonATCAircraft](#)
 - [SimConnect_AICreateParkedATCAircraft](#)
 - [SimConnect_AICreateSimulatedObjectEx](#)
 - [SimConnect_AIRemoveObject](#)
 - [SimConnect API Reference](#)
-

SimConnect_AICreateSimulatedObjectEx

The **SimConnect_AICreateSimulatedObjectEx** function is used to create AI controlled objects other than aircraft.

Syntax

```
HRESULT SimConnect_AICreateSimulatedObjectEx(  
    HANDLE hSimConnect,  
    const char* szContainertitle,  
    SIMCONNECT_DATA_INITPOSITION InitPos,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    BOOL isDisabled = FALSE  
    int OwnerID = -1  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szContainertitle

[in] Null-terminated string containing the container title. The container title is case-sensitive and can be found in the sim.cfg file, for example:

Object Types	Examples
--------------	----------

Ground Vehicles	title=Automobile title=FuelTruck title=VEH_air_bagcart1 title=VEH_air_bagcart_FlatNosed
Boats	title=cargoA title=cargoA_hoop title=cargoB
Miscellaneous	title=Flour_bomb_floating_dock title=HumpbackWhale

InitPos

[in] Specifies the initial position, using a [SIMCONNECT_DATA_INITPOSITION](#) structure.

RequestID

[in] Specifies the client defined request ID.

isDisabled

[in, optional] Specifies whether the simulation starts disabled or not default value is FALSE.

OwnerID

[in, optional] Specifies the owner of this simulated object. Default value is -1 which is an invalid object ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [AI Objects and Waypoints](#)
[Managed AI Waypoints](#)

Remarks

This function can be used to create a stationary aircraft (such as an unflyable aircraft on display outside a flight museum), but is typically intended to create simulation objects other than aircraft (such as ground vehicles, boats, and a number of special objects such as humpback whales and hot-air balloons). A number of errors apply to AI objects (refer to the [SIMCONNECT_EXCEPTION](#) enum for more details).

See Also

- [SimConnect_AICreateEnrouteATCAircraft](#)
- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateParkedATCAircraft](#)
- [SimConnect_AIRemoveObject](#)
- [SimConnect API Reference](#)

SimConnect_AIReleaseControl

The **SimConnect_AIReleaseControl** function is used to clear the AI control of a simulated object, typically an aircraft, in order for it to be controlled by a SimConnect client.

Syntax

```
HRESULT SimConnect_AIReleaseControl(
    HANDLE hSimConnect,
    SIMCONNECT_OBJECT_ID ObjectID,
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

ObjectID
[in] Specifies the server defined object ID.

RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

This function should be used to transfer the control of an aircraft, or other object, from the AI system to the SimConnect client. If this is not done the AI system and client may fight each other with unpredictable results. To prevent the simulation engine from updating the latitude, longitude, altitude and attitude of an aircraft, refer to the range of KEY_FREEZE..... [Event IDs](#).

The object ID can be obtained in a number of ways, refer to the [SimConnect_RequestDataOnSimObjectType](#) call, and also the use of the [SIMCONNECT_RECV_ASSIGNED_OBJECT_ID](#) structure.

See Also

- [SimConnect API Reference](#)
- [SimConnect_AICreateEnrouteATCAircraft](#)
- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateParkedATCAircraft](#)
- [SimConnect_AICreateSimulatedObject](#)
- [SimConnect_AISetAircraftFlightPlan](#)
- [SimConnect_AIReleaseControlEx](#)

SimConnect_AIReleaseControlEx

The **SimConnect_AIReleaseControlEx** function is used to clear the AI control of a simulated object, typically an aircraft, in order for it to be controlled by a SimConnect client. Optionally, this function can be used to completely remove the AI instead of clearing it. Removing the AI will allow for complete manual control of a SimConnect SimObject.

Syntax

```
HRESULT SimConnect_AIReleaseControlEx(  
    HANDLE hSimConnect,  
    SIMCONNECT_OBJECT_ID ObjectID,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    BOOL destroyAI = FALSE  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ObjectID

[in] Specifies the server defined object ID.

RequestID

[in] Specifies the client defined request ID.

destroyAI

[in, optional] Specifies whether to destroy the AI or not (default value = FALSE).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

This function should be used to transfer the control of an aircraft, or other object, from the AI system to the SimConnect client. If this is not done the AI system and client may fight each other with unpredictable results. To prevent the simulation engine from updating the latitude, longitude, altitude and attitude of an aircraft, refer to the range of KEY_FREEZE..... [Event IDs](#).

The object ID can be obtained in a number of ways, refer to the

[SimConnect_RequestDataOnSimObjectType](#) call, and also the use of the [SIMCONNECT_RECV_ASSIGNED_OBJECT_ID](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AICreateEnrouteATCAircraft](#)
 - [SimConnect_AICreateNonATCAircraft](#)
 - [SimConnect_AICreateParkedATCAircraft](#)
 - [SimConnect_AICreateSimulatedObject](#)
 - [SimConnect_AISetAircraftFlightPlan](#)
-

SimConnect_AIRemoveObject

The **SimConnect_AIRemoveObject** function is used to remove any object created by the client using one of the SimConnect AI creation functions.

Syntax

```
HRESULT SimConnect_AIRemoveObject(  
    HANDLE hSimConnect,  
    SIMCONNECT_OBJECT_ID ObjectID,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ObjectID

[in] Specifies the server defined object ID (refer to the [SIMCONNECT_RECV_ASSIGNED_OBJECT_ID](#) structure).

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

A client application can only remove AI controlled objects that it created, not objects created by other clients, or by the simulation itself.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AICreateEnrouteATCAircraft](#)
 - [SimConnect_AICreateNonATCAircraft](#)
 - [SimConnect_AICreateParkedATCAircraft](#)
 - [SimConnect_AICreateSimulatedObject](#)
-

SimConnect_AISetAircraftFlightPlan

The **SimConnect_AISetAircraftFlightPlan** function is used to set or change the flight plan of an AI controlled aircraft.

Syntax

```
HRESULT SimConnect_AISetAircraftFlightPlan(
    HANDLE hSimConnect,
    SIMCONNECT_OBJECT_ID ObjectID,
    const char* szFlightPlanPath,
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

ObjectID
[in] Specifies the server defined object ID.

szFlightPlanPath
[in] Null-terminated string containing the path to the flight plan file. Flight plans have the extension .pln, but no need to enter an extension here. The easiest way to create flight plans is to create them from within the simulation, and then save them off for use with the AI controlled aircraft. There is no need to enter the full path (just the filename) if the flight plan is in the default *Prepar3D v5 Files* directory. See the section on [Aircraft Flight Plans](#).

RequestID
[in] Specifies client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [AI Traffic](#)

Remarks

A number of errors, including **SIMCONNECT_EXCEPTION_CREATE_FLIGHTPLAN_FAILED**, apply to AI objects (refer to the [SIMCONNECT_EXCEPTION](#) enum for more details).

Typically this function would be used some time after the aircraft was created using the [SimConnect_AICreateParkedATCAircraft](#) call.

See Also

- [SimConnect API Reference](#)
- [SimConnect_AICreateEnrouteATCAircraft](#)
- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateParkedATCAircraft](#)

SimConnect_AISetAircraftFlightPlanW

The **SimConnect_AISetAircraftFlightPlanW** function is used to set or change the flight plan of an AI controlled aircraft. This version of the function supports unicode file paths.

Syntax

```
HRESULT SimConnect_AISetAircraftFlightPlanW(
    HANDLE hSimConnect,
    SIMCONNECT_OBJECT_ID ObjectID,
```

```
const wchar_t* szFlightPlanPath,
SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

ObjectID

[in] Specifies the server defined object ID.

szFlightPlanPath

[in] Null-terminated string containing the path to the flight plan file. Flight plans have the extension .pln, but no need to enter an extension here. The easiest way to create flight plans is to create them from within the simulation, and then save them off for use with the AI controlled aircraft.

There is no need to enter the full path (just the filename) if the flight plan is in the default *Prepar3D v5 Files* directory. See the section on [Aircraft Flight Plans](#).

RequestID

[in] Specifies client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [AI Traffic](#)

Remarks

A number of errors, including **SIMCONNECT_EXCEPTION_CREATE_FLIGHTPLAN_FAILED**, apply to AI objects (refer to the [SIMCONNECT_EXCEPTION](#) enum for more details).

Typically this function would be used some time after the aircraft was created using the [SimConnect_AICreateParkedATCAircraft](#) call.

See Also

- [SimConnect API Reference](#)
- [SimConnect_AICreateEnrouteATCAircraft](#)
- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateParkedATCAircraft](#)

SimConnect_AISetGroundClamp

The **SimConnect_AISetGroundClamp** function is used to enable/disable ground clamping on a SimConnect owned SimObject.

Syntax

```
HRESULT SimConnect_AISetGroundClamp(
    HANDLE hSimConnect,
    SIMCONNECT_OBJECT_ID ObjectID,
    BOOL bGroundClamp,
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
ObjectID
[in] Specifies the server defined object ID.
bGroundClamp
[in] Boolean value that specifies if Ground Clamping should be enabled or disabled for the specified ObjectID. Ground Clamping keeps the object from being pushed below ground level.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
- [SimConnect_AICreateEnrouteATCAircraft](#)
- [SimConnect_AICreateNonATCAircraft](#)
- [SimConnect_AICreateParkedATCAircraft](#)
- [SimConnect_AICreateSimulatedObject](#)
- [SimConnect_AICreateObjectWithExternalSim](#)

Attach Sim Specific Functions

Overview

The Attach Sim APIs are used to attach one SimObject to another SimObject. There's no physical simulation of the attached objects (i.e. the mass of an attached object doesn't affect the object it's attached to, the attached object just suspends simulation and moves with the object it's attached to). When an attached object is released, its primary sim resumes operation.

There are two versions of the attach API, [AttachObjectToSimObject](#) will create a new AI object and attach it to the specified SimObject. AttachSimObjectToSimObject is used to attach one existing SimObject to another SimObject (you can both attach the User SimObject to another object or attach other SimObjects to a User SimObject with this function).

SimConnect_AttachObjectToSimObject

The **SimConnect_AttachObjectToSimObject** function is used to create a new AI SimObject and attach it to an existing SimObject.

Syntax

```
HRESULT SimConnect_AttachObjectToSimObject(
    HANDLE hSimConnect,
    DWORD dwObjectID,
    SIMCONNECT_DATA_XYZ vecOff1,
    SIMCONNECT_DATA_PBH rotOff1,
    const char * szContainertitle
    SIMCONNECT_DATA_XYZ vecOff2
    SIMCONNECT_DATA_PBH rotOff2
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

dwObjectID
[in] ObjectID of the existing SimObject to attach to.

vecOff1
[in] XYZ offset from existing SimObject model center to attachment point.

rotOff1
[in] PBH offset from existing SimObject model body axis to attachment point.

szContainertitle
[in] Null-terminated string containing the container title of the AI SimObject you want to create and attach to the existing SimObject. The container title is found in the aircraft.cfg file (see the [Aircraft Configuration Files](#) document): for example: **title=Mooney Bravo** or **title=Commercial Airliner**.

vecOff2
[in] XYZ offset from created AI SimObject model center to attachment point.

rotOff2
[in] PBH offset from created AI SimObject model body axis to attachment point.

RequestID
[in] Specifies the ID of the client-defined request. This is used later by the client to identify which data has been received. This value should be unique for each request, re-using a *RequestID* will overwrite any previous request using the same ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_AttachSimObjectToSimObject](#)
- [SimConnect_ReleaseSimObjectFromSimObject](#)
- [SimConnect API Reference](#)

SimConnect_AttachSimObjectToSimObject

The **SimConnect_AttachSimObjectToSimObject** function is used to create a new AI SimObject and attach it to an existing SimObject.

Syntax

```
HRESULT SimConnect_AttachSimObjectToSimObject(
    HANDLE hSimConnect,
    DWORD dwObjectID1,
    SIMCONNECT_DATA_XYZ vecOff1,
    SIMCONNECT_DATA_PBH rotOff1,
    DWORD dwObjectID2
    SIMCONNECT_DATA_XYZ vecOff2
    SIMCONNECT_DATA_PBH rotOff2
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

dwObjectID1
[in] ObjectID of the existing SimObject to attach to.

vecOff1

[in] XYZ offset from dwObjectID1's model center to attachment point.
rotOff1
[in] PBH offset from dwObjectID1's model body axis to attachment point.
dwObjectID2
[in] ObjectID of the existing SimObject that is being attached to dwObjectID1.
vecOff2
[in] XYZ offset from dwObjectID2's model center to attachment point.
rotOff2
[in] PBH offset from dwObjectID2's model body axis to attachment point.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_AttachObjectToSimObject](#)
- [SimConnect_ReleaseSimObjectFromSimObject](#)
- [SimConnect API Reference](#)

SimConnect_ReleaseSimObjectFromSimObject

The **SimConnect_ReleaseSimObjectFromSimObject** function is used to release a SimObject from the SimObject it is attached to.

Syntax

```
HRESULT SimConnect_ReleaseSimObjectFromSimObject(  
    HANDLE hSimConnect,  
    DWORD dwObjectID1,  
    DWORD dwObjectID2  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
dwObjectID1
[in] ObjectID of the existing SimObject that dwObjectID2 is being released from.
dwObjectID2
[in] ObjectID of the existing SimObject that is being released from dwObjectID1.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_AttachObjectToSimObject](#)
 - [SimConnect_AttachSimObjectToSimObject](#)
 - [SimConnect API Reference](#)
-

Attach Weapon Specific Functions

Overview

The Attach Weapon APIs are used to attach one Weapon to a SimObject. The attach call can be made multiple times to fill out the required number of weapons. The hardpoints must be setup on the SimObject in order to use the Attach Weapon APIs.

SimConnect_AttachWeaponToObject

The **SimConnect_AttachWeaponToObject** function is used to attach a Weapon to a SimObject.

Syntax

```
HRESULT SimConnect_AttachWeaponToObject(  
    HANDLE hSimConnect,  
    const char* szContainertitle,  
    SIMCONNECT_OBJECT_ID dwObjectID  
    int attachPoint  
    int numOfRounds  
)
```

Weapon Examples

title=AIM-9M_Sidewinder
title=AGM-88_HARM
title=AIM-120_AMRAAM

Parameters

hSimConnect
 [in] Handle to a SimConnect object.
szContainertitle
 [in] null-terminated string containing title of the Weapon being attached to ObjectID.
dwObjectID
 [in] ObjectID of the existing SimObject.
attachPoint
 [in] int containing the station index from the object's Attachments.xml file.
numOfRounds
 [in] int containing the number of rounds of the weapon being attached.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

-
- [SimConnect_ClearWeapons](#)
 - [SimConnect API Reference](#)
-

SimConnect_ClearWeapons

The **SimConnect_ClearWeapons** function is used to remove all Weapons from a SimObject.

Syntax

```
HRESULT SimConnect_ClearWeapons(  
    HANDLE hSimConnect,  
    SIMCONNECT_OBJECT_ID dwObjectID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
dwObjectID
[in] ObjectID of the existing SimObject.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_AttachWeaponToObject](#)
 - [SimConnect API Reference](#)
-

External Sim Specific Functions

Overview

The External Sim APIs are used to write your own simulation code in a SimConnect Addon. When using an External Sim, instead of your SimObject behaving like an aircraft, helicopter, boat, ground vehicle, etc, it will behave how ever the code in your external sim add-on wants it to.

There are two primary ways to use/create an External Sim based SimObject:

1. Define a complete SimObject container that has a Category=ExternalSim line in the [General] section of the sim.cfg/aircraft.cfg file and at least an ExternalSimID = {GUID_STRING} line in the [fltsim.n] sections, for an External Secondary Sim, you just need an ExternalSecondarySimID = {GUID_STRING} in the [fltsim.r] section. When using this method you can use the standard Select Aircraft/Vehicle dialog to select the SimObject for use, or any of the standard AICreateXxx SimConnect functions. There is a [Sample External Sim SimObject container](#) included in the SDK that you can use as a reference/template for creating your own External Sim related SimObjects.
 2. Use the API functions that allow creating AI objects or User objects using an existing SimObject container but providing your own external sim code to run instead of whatever would normally run for that SimObject. When using this method, you must use one of the External Sim related create/attach functions - [SimConnect_AICreateObjectWithExternalSim](#), [SimConnect_ChangeVehicleWithExternalSim](#), or
-

[SimConnect_AttachExternalSecondarySimToSimObject](#).

There are two types of External Sims:

1. An External Sim which provides all of the required code to simulate the vehicles movement. This provides the primary sim code for the SimObject.
2. An External Secondary Sim which just provides additional simulation to another SimObject (the primary sim could be one of the built in categories, or another External Sim). You could have a tracked vehicle primary sim, and then use an External Secondary Sim to provide the simulation/drive animation vars/etc for a tank turret, and the same tank turret External Secondary Sim could also be used with a Wheeled vehicle primary sim or a Boat vehicle primary sim.

The External Sim APIs use the [Synchronous SimConnect](#) functionality implicitly (meaning you need to use one of the Unblock methods in all of the External Sim related callbacks). This is why the Register External Sim function accepts a callback mask, set this to only include the callbacks you actually implement handlers for.

The first thing an External Sim add-on needs to do is use the [RegisterExternalSim/RegisterExternalSecondarySim](#) function to let SimConnect know you want to provide External Sim services. To register, you provide a GUID (used to reference this external sim from sim.cfg/aircraft.cfg files or other API functions), a callback mask (set to only the callbacks you actually implement handlers for), and a structure definition ID of the data structure that will be passed to the Simulate callback.

Before your add-on code exits, it should use the [UnregisterExternalSim/UnregisterExternalSecondarySim](#) function to let SimConnect know you will no longer be providing services for that GUID.

There are 5 External Sim Callbacks:

1. [Create](#) - this is called whenever a SimObject using your External Sim is created.
2. [Destroy](#) - this is called whenever a SimObject using your External Sim is destroyed.
3. [Simulate](#) - this is called every simulation frame for each SimObject using your External Sim.
4. [Location Changed](#) - this is called whenever a SimObject has potentially moved, primarily due to user UI manipulations.
5. [Event](#) - this is called whenever a mapped event occurs. In order to map specific events be sent to your external sim add-on, you use the [MapClientEventToSimEvent](#) function.

To programmatically create AI objects that use a specific External Sim, you use the [AICreateObjectWithExternalSim](#) function to load an existing SimObject but override it to use an External Sim instead.

To programmatically create a User object that uses a specific External Sim, you use the [ChangeVehicleWithExternalSim](#) function to load an existing SimObject but override it to use an External Sim instead.

For External Secondary Sims, you don't use a create function, instead you use [AttachExternalSecondarySimToSimObject/ DetachExternalSecondarySimFromSimObject](#) to attach/detach an External Secondary sim to/from an existing Simobject.

A primary External Sim can write to any of the defined SimVars. A Secondary External Sim can write to any of the defined SimVars that aren't already being handled by the primary sim.

An External Sim can also make use of the [Ground Info APIs](#) to get a small grid of ground data around it's SimObject.

Besides providing the ability to simulate the physics for a SimObject, the External Sim APIs can also be used to create pseudo-multiplayer style add-ons allowing some external data source (an Entity Generator or another sim) to drive SimObjects around the sim world.

SimConnect_RegisterExternalSim

The **SimConnect_RegisterExternalSim** function is used to register a SimConnect client as an external sim provider.

Syntax

```
HRESULT SimConnect_RegisterExternalSim(  
    HANDLE hSimConnect,  
    const GUID guidExternalSimID,  
    SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG CallbackMask,  
    SIMCONNECT_DATA_DEFINITION_ID DefineID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidExternalSimID

[in] The GUID used to denote this external sim.

CallbackMask

[in] One or more members of the [SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG](#) enumeration that specify which External Sim Callbacks this client wants to receive.

DefineID

[in] The data definition ID for any data you would like to have sent with the External Sim Simulate Callback.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [External Sim](#)

See Also

- [SimConnect_UnregisterExternalSim](#)
- [SimConnect_RegisterExternalSecondarySim](#)
- [SimConnect_UnregisterExternalSecondarySim](#)
- [SimConnect_AttachExternalSecondarySimToSimObject](#)
- [SimConnect_DetachExternalSecondarySimFromSimObject](#)
- [SimConnect_AICreateObjectWithExternalSim](#)
- [SimConnect_ChangeVehicleWithExternalSim](#)
- [SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG](#)
- [SimConnect API Reference](#)

SimConnect_UnregisterExternalSim

The **SimConnect_UnregisterExternalSim** function is used to unregister a SimConnect client as an external sim provider.

Syntax

```
HRESULT SimConnect_UnregisterExternalSim()
```

```
HANDLE hSimConnect,
const GUID guidExternalSimID
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidExternalSimID

[in] The GUID used to denote this external sim.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [External Sim](#)

See Also

- [SimConnect_RegisterExternalSim](#)
- [SimConnect_RegisterExternalSecondarySim](#)
- [SimConnect_UnregisterExternalSecondarySim](#)
- [SimConnect_AttachExternalSecondarySimToSimObject](#)
- [SimConnect_DetachExternalSecondarySimFromSimObject](#)
- [SimConnect_AICreateObjectWithExternalSim](#)
- [SimConnect_ChangeVehicleWithExternalSim](#)
- [SimConnect API Reference](#)

SimConnect_RegisterExternalSecondarySim

The **SimConnect_RegisterExternalSecondarySim** function is used to register a SimConnect client as an external secondary sim provider.

Syntax

```
HRESULT SimConnect_RegisterExternalSecondarySim(
    HANDLE hSimConnect,
    const GUID guidExternalSecondarySimID,
    SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG CallbackMask,
    SIMCONNECT_DATA_DEFINITION_ID DefineID
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidExternalSecondarySimID

[in] The GUID used to denote this external secondary sim.

CallbackMask

[in] One or more members of the [SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG](#) enumeration that specify which External Sim Callbacks this client wants to receive.

DefinID

[in] The data definition ID for any data you would like to have sent with the External Sim Simulate Callback.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_RegisterExternalSim](#)
- [SimConnect_UnregisterExternalSim](#)
- [SimConnect_UnregisterExternalSecondarySim](#)
- [SimConnect_AttachExternalSecondarySimToSimObject](#)
- [SimConnect_DetachExternalSecondarySimFromSimObject](#)
- [SimConnect_AICreateObjectWithExternalSim](#)
- [SimConnect_ChangeVehicleWithExternalSim](#)
- [SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG](#)
- [SimConnect API Reference](#)

SimConnect_UnregisterExternalSecondarySim

The **SimConnect_UnregisterExternalSecondarySim** function is used to unregister a SimConnect client as an external secondary sim provider.

Syntax

```
HRESULT SimConnect_UnregisterExternalSecondarySim(  
    HANDLE hSimConnect,  
    const GUID guidExternalSecondarySimID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidExternalSecondarySimID

[in] The GUID used to denote this external secondary sim.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_RegisterExternalSim](#)
- [SimConnect_UnregisterExternalSim](#)

-
- [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect_AttachExternalSecondarySimToSimObject](#)
 - [SimConnect_DetachExternalSecondarySimFromSimObject](#)
 - [SimConnect_AICreateObjectWithExternalSim](#)
 - [SimConnect_ChangeVehicleWithExternalSim](#)
 - [SimConnect API Reference](#)
-

SimConnect_AttachExternalSecondarySimToSimObject

The **SimConnect_AttachExternalSecondarySimToSimObject** function is used to attach an External Secondary Sim to an existing SimObject.

Syntax

```
HRESULT SimConnect_AttachExternalSecondarySimToSimObject(  
    HANDLE hSimConnect,  
    DWORD dwObjectID,  
    const GUID guidExternalSecondarySimID,  
    const char * szExternalSimParams,  
    DWORD dwExternalSimVarCount  
) ;
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

dwObjectID

[in] ID of the SimObject to attach to.

guidExternalSecondarySimID

[in] The GUID of the external secondary sim you want to attach.

szExternalSimParams

[in] A string containing parameters to pass to the external secondary sim. Not used internally, but retrievable through the **EXTERNAL PRIMARY SIM DATA** simulation variable.

dwExternalSimVarCount

[in] Number of variables to allocate for this external secondary sim to use (range 0 to 999).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_UnregisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect_UnregisterExternalSecondarySim](#)
 - [SimConnect_DetachExternalSecondarySimFromSimObject](#)
 - [SimConnect_AICreateObjectWithExternalSim](#)
 - [SimConnect_ChangeVehicleWithExternalSim](#)
 - [SimConnect API Reference](#)
-

SimConnect_DetachExternalSecondarySimFromSimObject

The **SimConnect_DetachExternalSecondarySimFromSimObject** function is used to detach an External Secondary Sim from an existing SimObject.

Syntax

```
HRESULT SimConnect_DetachExternalSecondarySimFromSimObject(
    HANDLE hSimConnect,
    DWORD dwObjectID,
    const GUID guidExternalSecondarySimID,
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
dwObjectID
[in] ID of the SimObject to attach to.
guidExternalSecondarySimID
[in] The GUID of the external secondary sim you want to attach.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_RegisterExternalSim](#)
- [SimConnect_UnregisterExternalSim](#)
- [SimConnect_RegisterExternalSecondarySim](#)
- [SimConnect_UnregisterExternalSecondarySim](#)
- [SimConnect_AttachExternalSecondarySimToSimObject](#)
- [SimConnect_AICreateObjectWithExternalSim](#)
- [SimConnect_ChangeVehicleWithExternalSim](#)
- [SimConnect API Reference](#)

SimConnect_AICreateObjectWithExternalSim

The **SimConnect_AICreateObjectWithExternalSim** function is used to create an AI object and override the sim to use a specific external secondary sim.

Syntax

```
HRESULT SimConnect_AICreateObjectWithExternalSim(
    HANDLE hSimConnect,
    const char * szContainertitle,
    SIMCONNECT_DATA_INITPOSITION InitPos,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    const GUID guidExternalSimID,
    const char * szExternalSimParams,
    DWORD dwExternalSimVarCount
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.
szContainertitle
[in] Null-terminated string containing the container title.
InitPos
[in] Specifies the initial position, using a [SIMCONNECT_DATA_INITPOSITION](#) structure.
RequestID
[in] Specifies the client defined request ID.
guidExternalSimID
[in] The GUID of the external secondary sim you want to use to override the default SimObject's sim.
szExternalSimParams
[in] A string containing parameters to pass to the external sim. Not used internally, but retrievable through the **EXTERNAL PRIMARY SIM DATA** simulation variable.
dwExternalSimVarCount
[in] Number of variables to allocate for this external sim to use (range 0 to 999).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_RegisterExternalSim](#)
- [SimConnect_UnregisterExternalSim](#)
- [SimConnect_RegisterExternalSecondarySim](#)
- [SimConnect_UnregisterExternalSecondarySim](#)
- [SimConnect_AttachExternalSecondarySimToSimObject](#)
- [SimConnect_DetachExternalSecondarySimFromSimObject](#)
- [SimConnect_ChangeVehicleWithExternalSim](#)
- [SimConnect API Reference](#)

SimConnect_ChangeVehicleWithExternalSim

The **SimConnect_ChangeVehicleWithExternalSim** function is used to load a new user vehicle and override the sim using a specific external sim.

Syntax

```
HRESULT SimConnect_ChangeVehicleWithExternalSim(  
    HANDLE hSimConnect,  
    const char * szContainertitle,  
    const GUID guidExternalSimID,  
    const char * szExternalSimParams,  
    DWORD dwExternalSimVarCount  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szContainertitle
[in] Null-terminated string containing the container title.
guidExternalSimID
[in] The GUID of the external secondary sim you want to use to override the default SimObject's sim.

szExternalSimParams

[in] A string containing parameters to pass to the external sim. Not used internally, but retrievable through the **EXTERNAL PRIMARY SIM DATA** simulation variable.

dwExternalSimVarCount

[in] Number of variables to allocate for this external sim to use (range 0 to 999).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
--------------	-------------

S_OK The function succeeded.

E_FAIL The function failed.

See Also

- [SimConnect_RegisterExternalSim](#)
- [SimConnect_UnregisterExternalSim](#)
- [SimConnect_RegisterExternalSecondarySim](#)
- [SimConnect_UnregisterExternalSecondarySim](#)
- [SimConnect_AttachExternalSecondarySimToSimObject](#)
- [SimConnect_DetachExternalSecondarySimFromSimObject](#)
- [SimConnect_AICreateObjectWithExternalSim](#)
- [SimConnect API Reference](#)

[- top -](#)

View Functions

Overview

To view a list of all view SimConnect functions, see the [View Functions](#) table.

SimConnect_CameraSetRelative6DOF

The **SimConnect_CameraSetRelative6DOF** function is used to adjust the user's aircraft view camera.

Syntax

```
HRESULT SimConnect_CameraSetRelative6DOF(  
    HANDLE hSimConnect,  
    float fDeltaX,  
    float fDeltaY,  
    float fDeltaZ,  
    float fPitchDeg,  
    float fBankDeg,  
    float fHeadingDeg  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

fDeltaX

[in] Float containing the delta in the x-axis from the eyepoint reference point. See the [\[views\]](#) section of the [Aircraft Configuration Files](#) document for a description of the eyepoint.

fDeltaY

[in] Float containing the delta in the y-axis from the eyepoint reference point.

fDeltaZ

[in] Float containing the delta in the z-axis from the eyepoint reference point.

fPitchDeg

[in] Float containing the pitch in degrees (rotation about the x axis). A positive value points the nose down, a negative value up. The range of allowable values is +90 to -90 degrees.

fBankDeg

[in] Float containing the bank angle in degrees (rotation about the z axis). The range of allowable values is +180 to -180 degrees.

fHeadingDeg

[in] Float containing the heading in degrees (rotation about the y axis). A positive value rotates the view right, a negative value left. If the user is viewing the 2D cockpit, the view will change to the Virtual Cockpit 3D view if the angle exceeds 45 degrees from the view ahead. The Virtual Cockpit view will change back to the 2D cockpit view if the heading angle drops below 45 degrees. The range of allowable values is +180 to -180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [Cockpit Camera](#)

Remarks

Any one of the six parameters can be set to **SIMCONNECT_CAMERA_IGNORE_FIELD** which indicates that the value for the camera should be taken unmodified from the reference point.

See Also

- [SimConnect API Reference](#)
-

SimConnect_CameraSetRelative6DofByName

The **SimConnect_CameraSetRelative6DofByName** function is used to adjust the specified view's camera.

Syntax

```
HRESULT SimConnect_CameraSetRelative6DofByName(
    HANDLE hSimConnect,
    const char* szName,
    float fDeltaX,
    float fDeltaY,
    float fDeltaZ,
    float fPitchDeg,
    float fBankDeg,
    float fHeadingDeg
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

fDeltaX

[in] Float containing the delta in the x-axis from the eyepoint reference point. See the [views] section of the [Aircraft Configuration Files](#) document for a description of the eyepoint.

fDeltaY

[in] Float containing the delta in the y-axis from the eyepoint reference point.

fDeltaZ

[in] Float containing the delta in the z-axis from the eyepoint reference point.

fPitchDeg

[in] Float containing the pitch in degrees (rotation about the x axis). A positive value points the nose down, a negative value up. The range of allowable values is +90 to -90 degrees.

fBankDeg

[in] Float containing the bank angle in degrees (rotation about the z axis). The range of allowable values is +180 to -180 degrees.

fHeadingDeg

[in] Float containing the heading in degrees (rotation about the y axis). A positive value rotates the view right, a negative value left. If the user is viewing the 2D cockpit, the view will change to the Virtual Cockpit 3D view if the angle exceeds 45 degrees from the view ahead. The Virtual Cockpit view will change back to the 2D cockpit view if the heading angle drops below 45 degrees. The range of allowable values is +180 to -180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Any one of the six parameters can be set to **SIMCONNECT_CAMERA_IGNORE_FIELD** which indicates that the value for the camera should be taken unmodified from the reference point.

See Also

- [SimConnect API Reference](#)
-

SimConnect_SetCameraSensorMode

The **SimConnect_SetCameraSensorMode** function is used to change a specific camera's sensor mode.

Syntax

```
HRESULT SimConnect_SetCameraSensorMode(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_CAMERA_SENSOR_MODE eSensorMode  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera. This should equal the camera definition name.

eSensorMode

[in] The sensor mode.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_CAMERA_SENSOR_MODE](#)
-

SimConnect_SetMainCameraSensorMode

The **SimConnect_SetMainCameraSensorMode** function is used to change the main camera's sensor mode.

Syntax

```
HRESULT SimConnect_SetMainCameraSensorMode(  
    HANDLE hSimConnect,  
    SIMCONNECT_CAMERA_SENSOR_MODE eSensorMode  
>;
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
eSensorMode
[in] The sensor mode.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_CAMERA_SENSOR_MODE](#)
-

SimConnect_CreateCameraDefinition

The **SimConnect_CreateCameraDefinition** function is used to create a new camera definition.

Syntax

```
HRESULT SimConnect_CreateCameraDefinition(  
    HANDLE hSimConnect,  
    GUID guidCamera,  
    SIMCONNECT_CAMERA_TYPE szViewType,  
    const char* sztitle,  
    SIMCONNECT_DATA_XYZ xyzBias,  
    SIMCONNECT_DATA_PBH pbhBias  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidCamera

[in] GUID a unique identifier for a new camera defintion.

szViewType

[in] The type of view to create. See [SIMCONNECT_CAMERA_TYPE](#).

sztitle

[in] Null-terminated string containing the title of the camera.

xyzBias

[in] SIMCONNECT_DATA_XYZ containing the offset from the origin.

pbhBias

[in] SIMCONNECT_DATA_PBH containing the orientation of the camera.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Camera System](#)

See Also

- [SimConnect API Reference](#)
- [SimConnect_DeleteCameraInstance](#)
- [SimConnect_CreateCameraInstance](#)
- [SIMCONNECT_CAMERA_TYPE](#)

SimConnect_CreateCameraInstance

The **SimConnect_CreateCameraInstance** function is used to create a new camera instance based on a camera definition.

Syntax

```
HRESULT SimConnect_CreateCameraInstance(  
    HANDLE hSimConnect,  
    GUID guidCamera,  
    const char* szName,  
    SIMCONNECT_OBJECT_ID dwObjectID,  
    SIMCONNECT_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidCamera

[in] GUID a unique identifier of a already created camera defintion.

szName

[in] Null-terminated string containing the name of the camera instance.

dwObjectID

[in] SIMCONNECT_OBJECT_ID containing the objectID of the SimObject to attach to the camera. Used with [SIMCONNECT_CAMERA_TYPE](#)

[SIMCONNECT_CAMERA_TYPE_OBJECT_AI_VIRTUAL](#).

RequestID

[in] SIMCONNECT_REQUEST_ID containing the request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Camera System](#)

See Also

- [SimConnect API Reference](#)
- [SimConnect_CreateCameraDefinition](#)
- [SimConnect_DeleteCameraInstance](#)
- [SIMCONNECT_CAMERA_TYPE](#)

SimConnect_DeleteCameraInstance

The **SimConnect_DeleteCameraInstance** function is used to remove a camera instance.

Syntax

```
HRESULT SimConnect_DeleteCameraInstance(
    HANDLE hSimConnect,
    GUID guidCamera,
    UINT32 instanceId
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidCamera

[in] GUID a unique identifier of a already created camera defintion.

instanceId

[in] UINT32 containing the camera instance identifier.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CreateCameraDefinition](#)
 - [SimConnect_CreateCameraInstance](#)
 - [SIMCONNECT_CAMERA_TYPE](#)
-

SimConnect_SetCameraHorizontalFov

The **SimConnect_SetCameraHorizontalFov** function is used to adjust the specified camera's horizontal field of view.

Syntax

```
HRESULT SimConnect_SetCameraHorizontalFov(
    HANDLE hSimConnect,
    const char* szName,
    float hFov
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera. This can either match the window title or the name of the camera

definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

hFov

[in] Float containing the new horizontal FoV. The range of allowable values is between 0 and 180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetCameraVerticalFov](#)
 - [SimConnect_SetCameraFov](#)
-

SimConnect_SetCameraVerticalFov

The **SimConnect_SetCameraVerticalFov** function is used to adjust the specified camera's vertical field of view.

Syntax

```
HRESULT SimConnect_SetCameraHorizontalFov(  
    HANDLE hSimConnect,  
    const char* szName,  
    float vFov  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

vFov

[in] Float containing the new vertical FoV. The range of allowable values is between 0 and 180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetCameraHorizontalFov](#)
 - [SimConnect_SetCameraFov](#)
-

SimConnect_SetCameraFov

The **SimConnect_SetCameraFov** function is used to adjust the specified camera's field of view.

Syntax

```
HRESULT SimConnect_SetCameraFov(
    HANDLE hSimConnect,
    const char* szName,
    float hFov,
    float vFov
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

hFov

[in] Float containing the new horizontal FoV. The range of allowable values is between 0 and 180 degrees.

vFov

[in] Float containing the new vertical FoV. The range of allowable values is between 0 and 180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
-

-
- [SimConnect_SetCameraHorizontalFov](#)
 - [SimConnect_SetCameraVerticalFov](#)
-

SimConnect_SetMainCameraHorizontalFov

The **SimConnect_SetMainCameraHorizontalFov** function is used to adjust the main camera's horizontal field of view.

Syntax

```
HRESULT SimConnect_SetMainCameraHorizontalFov(  
    HANDLE hSimConnect,  
    float hFov  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

hFov

[in] Float containing the new horizontal FoV. The range of allowable values is between 0 and 180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetMainCameraVerticalFov](#)
 - [SimConnect_SetMainCameraFov](#)
-

SimConnect_SetMainCameraVerticalFov

The **SimConnect_SetMainCameraVerticalFov** function is used to adjust the main camera's vertical field of view.

Syntax

```
HRESULT SimConnect_SetMainCameraVerticalFov(  
    HANDLE hSimConnect,  
    float vFov  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

vFov

[in] Float containing the new vertical FoV. The range of allowable values is between 0 and 180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetMainCameraHorizontalFov](#)
 - [SimConnect_SetMainCameraFov](#)
-

SimConnect_SetMainCameraFov

The **SimConnect_SetMainCameraFov** function is used to adjust the main camera's field of view.

Syntax

```
HRESULT SimConnect_SetMainCameraFov(  
    HANDLE hSimConnect,  
    float hFov,  
    float vFov  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

hFov

[in] Float containing the new horizontal FoV. The range of allowable values is between 0 and 180 degrees.

vFov

[in] Float containing the new vertical FoV. The range of allowable values is between 0 and 180 degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetMainCameraHorizontalFov](#)
 - [SimConnect_SetMainCameraVerticalFov](#)
-

SimConnect_CameraZoomIn

The **SimConnect_CameraZoomIn** function is used to adjust the zoom of the specified camera.

Syntax

```
HRESULT SimConnect_CameraZoomIn(  
    HANDLE hSimConnect,  
    const char* szName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CameraZoomOut](#)
-

SimConnect_CameraZoomOut

The **SimConnect_CameraZoomOut** function is used to adjust the zoom of the specified camera.

Syntax

```
HRESULT SimConnect_CameraZoomOut(  
    HANDLE hSimConnect,  
    const char* szName
```

);

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CameraZoomIn](#)
-

SimConnect_MainCameraZoomIn

The **SimConnect_MainCameraZoomIn** function is used to adjust the zoom of the main camera.

Syntax

```
HRESULT SimConnect_MainCameraZoomIn(  
    HANDLE hSimConnect  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

-
- [SimConnect API Reference](#)
 - [SimConnect_MainCameraZoomOut](#)
-

SimConnect_MainCameraZoomOut

The **SimConnect_MainCameraZoomOut** function is used to adjust the zoom of the main camera.

Syntax

```
HRESULT SimConnect_MainCameraZoomOut(  
    HANDLE hSimConnect  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_MainCameraZoomIn](#)
-

SimConnect_ChangeView

The **SimConnect_ChangeView** function is used to change the camera on the main view.

Syntax

```
HRESULT SimConnect_ChangeView(  
    HANDLE hSimConnect,  
    const char* szName  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szName
[in] The name of the camera to switch to.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_OpenView](#)
 - [SimConnect_CloseView](#)
-

SimConnect_OpenView

The **SimConnect_OpenView** function is used to open a new view.

Syntax

```
HRESULT SimConnect_OpenView(  
    HANDLE hSimConnect,  
    const char* szName  
    const char* szTitle = 0  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the view to open.

szTitle

[in, optional] The title of the window to open (defaults to NULL).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Camera System](#)

See Also

-
- [SimConnect API Reference](#)
 - [SimConnect_ChangeView](#)
 - [SimConnect_CloseView](#)
-

SimConnect_CloseView

The **SimConnect_CloseView** function is used to close a view.

Syntax

```
HRESULT SimConnect_CloseView(  
    HANDLE hSimConnect,  
    const char* szName  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szName
[in] The name of the view to close.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_ChangeView](#)
 - [SimConnect_OpenView](#)
-

SimConnect_UndockView

The **SimConnect_UndockView** function is used to undock a view.

Syntax

```
HRESULT SimConnect_UndockView(  
    HANDLE hSimConnect,  
    const char* szName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.
szName
[in] The name of the view to undock.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Camera System](#)

See Also

- [SimConnect API Reference](#)
 - [SimConnect_DockView](#)
-

SimConnect_DockView

The **SimConnect_DockView** function is used to dock a view.

Syntax

```
HRESULT SimConnect_DockView(  
    HANDLE hSimConnect,  
    const char* szName  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szName
[in] The name of the view to dock.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Camera System](#)

See Also

- [SimConnect API Reference](#)
 - [SimConnect_UndockView](#)
-

SimConnect_SetCameraWindowPosition

The **SimConnect_SetCameraWindowPosition** function is used to set the window position of a view.

Syntax

```
HRESULT SimConnect_SetCameraWindowPosition(  
    HANDLE hSimConnect,  
    const char* szName,  
    UINT uX,  
    UINT uY  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the view to move.

uX

[in] New x position of the view.

uY

[in] New y position of the view.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Camera System](#)

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetCameraWindowSize](#)
-
-

SimConnect_SetCameraWindowSize

The **SimConnect_SetCameraWindowSize** function is used to set the window size of a view.

Syntax

```
HRESULT SimConnect_SetCameraWindowSize(  
    HANDLE hSimConnect,  
    const char* szName,  
    UINT uWidth,  
    UINT uHeight  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szName
[in] The name of the view to size.
uWidth
[in] New width of the view.
uHeight
[in] New height of the view.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Camera System](#)

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetCameraWindowPosition](#)
-

SimConnect_AddPostProcess

The **SimConnect_AddPostProcess** function is used to apply a post process effect to the specified camera.

Syntax

```
HRESULT SimConnect_AddPostProcess(  
    HANDLE hSimConnect,  
    const char* szCameraName,
```

```
const char* szPostProcessName  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szCameraName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

szPostProcessName

[in] The name of the post process effect.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AddPostProcessMainCamera](#)
 - [SimConnect_RemovePostProcess](#)
 - [SimConnect_RemovePostProcessMainCamera](#)
-

SimConnect_AddPostProcessMainCamera

The **SimConnect_AddPostProcessMainCamera** function is used to apply a post process effect to the main camera.

Syntax

```
HRESULT SimConnect_AddPostProcessMainCamera(  
    HANDLE hSimConnect,  
    const char* szPostProcessName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szPostProcessName

[in] The name of the post process effect.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AddPostProcess](#)
 - [SimConnect_RemovePostProcess](#)
 - [SimConnect_RemovePostProcessMainCamera](#)
-

SimConnect_RemovePostProcess

The **SimConnect_RemovePostProcess** function is used to remove a post process effect from the specified camera.

Syntax

```
HRESULT SimConnect_RemovePostProcess(  
    HANDLE hSimConnect,  
    const char* szCameraName,  
    const char* szPostProcessName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szCameraName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

szPostProcessName

[in] The name of the post process effect.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AddPostProcess](#)
-

-
- [SimConnect_AddPostProcessMainCamera](#)
 - [SimConnect_RemovePostProcessMainCamera](#)
-

SimConnect_RemovePostProcessMainCamera

The **SimConnect_RemovePostProcessMainCamera** function is used to remove a post process effect from the main camera.

Syntax

```
HRESULT SimConnect_RemovePostProcessMainCamera(  
    HANDLE hSimConnect,  
    const char* szPostProcessName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szPostProcessName

[in] The name of the post process effect.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_AddPostProcess](#)
 - [SimConnect_RemovePostProcessMainCamera](#)
 - [SimConnect_RemovePostProcess](#)
-

SimConnect_CameraSmoothRelative6DOF

The **SimConnect_CameraSmoothRelative6DOF** function is used to smoothly transition the view of the main camera to the specified location.

Syntax

```
HRESULT SimConnect_CameraSmoothRelative6DOF(  
    HANDLE hSimConnect,  
    float fDeltaX,  
    float fDeltaY,  
    float fDeltaZ,  
    float fPitchDeg,  
    float fBankDeg,
```

```
    float fHeadingDeg,  
    float fSmoothPanTime = 0  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

fDeltaX

[in] The new x position of the camera relative to the origin in meters. Valid range: -100.0 - 100.0.

fDeltaY

[in] The new y position of the camera relative to the origin in meters. Valid range: -100.0 - 100.0.

fDeltaZ

[in] The new z position of the camera relative to the origin in meters. Valid range: -100.0 - 100.0.

fPitchDeg

[in] The new pitch of the camera relative to the origin in degrees. Valid range: -90.0 - 90.0.

fBankDeg

[in] The new bank of the camera relative to the origin in degrees. Valid range: -90.0 - 90.0.

fHeadingDeg

[in] The new heading of the camera relative to the origin in degrees. Valid range: -90.0 - 90.0.

fSmoothPanTime

[in, optional] The camera pan duration. If no value is specified the value defined in the

Cameras.cfg will be used (defaults to 1.0f).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
- [SimConnect_CameraSmoothRelative6DofByName](#)
- [SimConnect_CameraPanToView](#)
- [SimConnect_MainCameraPanToView](#)

SimConnect_CameraSmoothRelative6DofByName

The **SimConnect_CameraSmoothRelative6DofByName** function is used to smoothly transition the view of the specified camera to the specified location.

Syntax

```
HRESULT SimConnect_CameraSmoothRelative6DofByName(  
    HANDLE hSimConnect,  
    const char* szCameraName,  
    float fDeltaX,  
    float fDeltaY,  
    float fDeltaZ,
```

```
    float fPitchDeg,
    float fBankDeg,
    float fHeadingDeg,
    float fSmoothPanTime = 0
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szCameraName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

fDeltaX

[in] The new x position of the camera relative to the origin in meters. Valid range: -100.0 - 100.0.

fDeltaY

[in] The new y position of the camera relative to the origin in meters. Valid range: -100.0 - 100.0.

fDeltaZ

[in] The new z position of the camera relative to the origin in meters. Valid range: -100.0 - 100.0.

fPitchDeg

[in] The new pitch of the camera relative to the origin in degrees. Valid range: -90.0 - 90.0.

fBankDeg

[in] The new bank of the camera relative to the origin in degrees. Valid range: -90.0 - 90.0.

fHeadingDeg

[in] The new heading of the camera relative to the origin in degrees. Valid range: -90.0 - 90.0.

fSmoothPanTime

[in, optional] The camera pan duration. If no value is specified the value defined in the Cameras.cfg will be used (defaults to 1.0f).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CameraSmoothRelative6DOF](#)
 - [SimConnect_CameraPanToView](#)
 - [SimConnect_MainCameraPanToView](#)
-

SimConnect_CameraPanToView

The **SimConnect_CameraPanToView** function is used to smoothly transition the view of the specified camera to the location of the specified target view. This function currently only works for [Custom Cameras](#).

Syntax

```
HRESULT SimConnect_CameraPanToView(
    HANDLE hSimConnect,
    const char* szCameraName,
    const char* szTargetName,
    float fSmoothPanTime = 0
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szCameraName

[in] The name of the camera. This can either match the window title or the name of the camera definition. If the window title is used only the camera for that window will be modified. If the camera definition name is used and more than one window is open corresponding to that definition than all of those windows will be modified.

szTargetName

[in] The name of the target view.

fSmoothPanTime

[in, optional] The camera pan duration. If no value is specified the value defined in the Cameras.cfg will be used (defaults to 1.0f).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CameraSmoothRelative6DOF](#)
 - [SimConnect_CameraSmoothRelative6DofByName](#)
 - [SimConnect_MainCameraPanToView](#)
-

SimConnect_MainCameraPanToView

The **SimConnect_MainCameraPanToView** function is used to smoothly transition the view of the main camera to the location of the specified target view. This function currently only works for [Custom Cameras](#).

Syntax

```
HRESULT SimConnect_MainCameraPanToView(
    HANDLE hSimConnect,
    const char* szTargetName,
    float fSmoothPanTime = 0
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szTargetName

[in] The name of the target view.

fSmoothPanTime

[in, optional] The camera pan duration. If no value is specified the value defined in the Cameras.cfg will be used (defaults to 1.0f).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CameraSmoothRelative6DOF](#)
 - [SimConnect_CameraSmoothRelative6DofByName](#)
 - [SimConnect_CameraPanToView](#)
-

SimConnect_SendCameraCommand

The **SimConnect_SendCameraCommand** function is used to simulate user inputs to control camera movement and rotation.

Syntax

```
HRESULT SimConnect_SendCameraCommand(  
    HANDLE hSimConnect,  
    cconst GUID guidCamera,  
    SIMCONNECT_CAMERA_COMMAND eCommand  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidCamera

[in] The GUID of the camera to move/rotate.

eCommand

[in] The [SIMCONNECT_CAMERA_COMMAND](#) that defines which user input is simulated.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the

following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_CAMERA_COMMAND](#)
-

SimConnect_RequestCameraFov

The **SimConnect_RequestCameraFov** function is used to request the field-of-view of the given camera.

Syntax

```
HRESULT SimConnect_RequestCameraFov(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szName
[in] The name of the camera to request data from.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestMainCameraFov](#)
 - [SimConnect_SetCameraFov](#)
 - [SimConnect_SetMainCameraFov](#)
-

SimConnect_RequestCameraRelative6DOF

The **SimConnect_RequestCameraRelative6DOF** function is used to request the XYZ delta offset from the eyepoint reference point in meters, as well as the pitch, bank, and heading in degrees of the main view.

Syntax

```
HRESULT SimConnect_RequestCameraRelative6DOF(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CameraSetRelative6DOF](#)
 - [SimConnect_CameraSetRelative6DofByName](#)
 - [SimConnect_RequestCameraRelative6DofByName](#)
-

SimConnect_RequestCameraRelative6DofByName

The **SimConnect_RequestCameraRelative6DofByName** function is used to request the XYZ delta offset from the eyepoint reference point in meters, as well as the pitch, bank, and heading in degrees of a given view.

Syntax

```
HRESULT SimConnect_RequestCameraRelative6DofByName(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

szName
[in] The name of the camera to request data from.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CameraSetRelative6DOF](#)
 - [SimConnect_CameraSetRelative6DofByName](#)
 - [SimConnect_RequestCameraRelative6DOF](#)
-

SimConnect_RequestCameraSensorMode

The **SimConnect_RequestCameraSensorMode** function is used to request the current sensor mode of a given view.

Syntax

```
HRESULT SimConnect_RequestCameraSensorMode(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szName
[in] The name of the camera to request data from.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestMainCameraSensorMode](#)
 - [SimConnect_SetCameraSensorMode](#)
 - [SimConnect_SetMainCameraSensorMode](#)
-

SimConnect_RequestCameraWindowPosition

The **SimConnect_RequestCameraWindowPosition** function is used to request the current window position of a given view.

Syntax

```
HRESULT SimConnect_RequestCameraWindowPosition(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the camera to request data from.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestCameraWindowSize](#)
 - [SimConnect_SetCameraWindowPosition](#)
 - [SimConnect_SetCameraWindowSize](#)
-

SimConnect_RequestCameraWindowSize

The **SimConnect_RequestCameraWindowSize** function is used to request the current window size of a given view.

Syntax

```
HRESULT SimConnect_RequestCameraWindowSize(
    HANDLE hSimConnect,
    const char* szName,
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
szName
[in] The name of the camera to request data from.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestCameraWindowPosition](#)
 - [SimConnect_SetCameraWindowPosition](#)
 - [SimConnect_SetCameraWindowSize](#)
-

SimConnect_RequestMainCameraFov

The **SimConnect_RequestMainCameraFov** is used to request the field-of-view of the main view (horizontal and vertical).

Syntax

```
HRESULT SimConnect_RequestMainCameraFov(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestCameraFov](#)
 - [SimConnect_SetCameraFov](#)
 - [SimConnect_SetMainCameraFov](#)
-

SimConnect_RequestMainCameraSensorMode

The **SimConnect_RequestMainCameraSensorMode** function is used to request the current sensor mode of the main view.

Syntax

```
HRESULT SimConnect_RequestMainCameraSensorMode(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestCameraSensorMode](#)
 - [SimConnect_SetCameraSensorMode](#)
 - [SimConnect_SetMainCameraSensorMode](#)
-

SimConnect_CaptureImage

The **SimConnect_CaptureImage** function is used to capture the image of a specific view.

Syntax

```
HRESULT SimConnect_CaptureImage(  
    HANDLE hSimConnect,  
    const char* szFileName,  
    const char* szFilePath,  
    UINT ulImageFormat,  
    const char* szViewName  
) ;
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] The name of the file to store image.

szFilePath

[in] The path of the file, can be relative or absolute.

ulImageFormat

[in] Specifies the format of the image. Formats: 0 BMP, 1 PNG, 2 JPEG, 3 TIFF.

szViewName

[in] The name of the window to capture the image.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_OpenView](#)
- [SimConnect_CloseView](#)
- [SimConnect_ChangeView](#)

SimConnect_CaptureImageW

The **SimConnect_CaptureImageW** function is used to capture the image of a specific view. This version of the function supports unicode file paths, filenames, and view names.

Syntax

```
HRESULT SimConnect_CaptureImageW(  
    HANDLE hSimConnect,  
    const wchar_t* szFileName,  
    const wchar_t* szFilePath,  
    UINT ulImageFormat,  
    const wchar_t* szViewName
```

);

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] The name of the file to store image.

szFilePath

[in] The path of the file, can be relative or absolute.

ulImageFormat

[in] Specifies the format of the image. Formats: 0 BMP, 1 PNG, 2 JPEG, 3 TIFF.

szViewName

[in] The name of the window to capture the image.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_OpenView](#)
- [SimConnect_CloseView](#)
- [SimConnect_ChangeView](#)

SimConnect_BeginVideoStream

The **SimConnect_BeginVideoStream** function is used to begin network streaming video of a specific view.

Syntax

```
SimConnect_BeginVideoStream(HANDLE hSimConnect, const char * szViewName,  
SIMCONNECT_DATA_VIDEO_STREAM_INFO StreamInfo) HRESULT  
SimConnect_BeginVideoStream(  
    HANDLE hSimConnect,  
    const char * szViewName,  
    SIMCONNECT_DATA_VIDEO_STREAM_INFO StreamInfo  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szViewName

[in] Name of view. If null, default or active view will be used.

StreamInfo

[in] Struct containing information about the video stream

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SIMCONNECT_DATA_VIDEO_STREAM_INFO](#)
 - [SimConnect_EndVideoStream](#)
 - [SimConnect_OpenView](#)
 - [SimConnect_CloseView](#)
 - [SimConnect_ChangeView](#)
-

SimConnect_EndVideoStream

The **SimConnect_EndVideoStream** function is used to end network streaming video of a specific view.

Syntax

```
HRESULT SimConnect_EndVideoStream(  
    HANDLE hSimConnect,  
    const char * szViewName,  
    SIMCONNECT_DATA_VIDEO_STREAM_INFO StreamInfo  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szViewName

[in] Name of view. If null, default or active view will be used.

StreamInfo

[in] Struct containing information about the video stream

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

-
- [**SIMCONNECT_DATA_VIDEO_STREAM_INFO**](#)
 - [**SimConnect_BeginVideoStream**](#)
 - [**SimConnect_OpenView**](#)
 - [**SimConnect_CloseView**](#)
 - [**SimConnect_ChangeView**](#)
-

Observer Specific Functions

SimConnect_CreateObserver

The **SimConnect_CreateObserver** function creates an observer using **SIMCONNECT_DATA_OBSERVER**.

Syntax

```
HRESULT SimConnect_CreateObserver(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_OBSERVER ObserverData,  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] Specifies a descriptive name for the new observer view. This name will appear on the dialogs of *Prepar3D*.

ObserverData

[in] Specifies all data for the observer to be created.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [**SimConnect_RequestObserverData**](#)
 - [**SimConnect_MoveObserver**](#)
 - [**SimConnect_RotateObserver**](#)
 - [**SimConnect_SetObserverPosition**](#)
-

-
- [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverLookAt](#)
 - [SimConnect_SetObserverLookAtEx](#)
 - [SimConnect_ObserverTrackLocationOn](#)
 - [SimConnect_SetObserverFieldOfView](#)
 - [SimConnect_SetObserverStepSize](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
 - [SimConnect_SetObserverRegime](#)
 - [SimConnect_SetObserverZoomLevels](#)
 - [SimConnect_ObserverTrackEntityOn](#)
 - [SimConnect_ObserverTrackEntityOff](#)
-

SimConnect_RequestObserverData

The **SimConnect_RequestObserverData** function is used to retrieve information about an observer.

Syntax

```
HRESULT SimConnect_RequestObserverData(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    const char* szName,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which data has been received. This value should be unique for each request, re-using a RequestID will overwrite any previous request using the same ID.

szName

[in] Name of the observer view that data is being requested.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

This function will return a [SIMCONNECT_RECV_OBSERVER_DATA](#) structure.

See Also

-
- [SimConnect_CreateObserver](#)
 - [SimConnect_MoveObserver](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverPosition](#)
 - [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverLookAt](#)
 - [SimConnect_SetObserverLookAtEx](#)
 - [SimConnect_ObserverTrackLocationOn](#)
 - [SimConnect_SetObserverFieldOfView](#)
 - [SimConnect_SetObserverStepSize](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
 - [SimConnect_SetObserverRegime](#)
 - [SimConnect_SetObserverZoomLevels](#)
 - [SimConnect_ObserverTrackEntityOn](#)
 - [SimConnect_ObserverTrackEntityOff](#)
-

SimConnect_MoveObserver

The **SimConnect_MoveObserver** function moves an observer using an XYZ translation.

Syntax

```
HRESULT SimConnect_MoveObserver(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_XYZ Translation,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

Translation

[in] The xyz translation used to move the observer.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_SetObserverPosition](#)
-

SimConnect_RotateObserver

The **SimConnect_RotateObserver** function rotates the specified observer view.

Syntax

```
HRESULT SimConnect_RotateObserver(  
    HANDLE hSimConnect,  
    const char* szName,  
    DWORD dwAxis,  
    double dAngleDegrees,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

dwAxis

[in] The axis to rotate around. *SIMCONNECT_OBSERVER_AXIS_PITCH* is 0, *SIMCONNECT_OBSERVER_AXIS_ROLL* is 1, and *SIMCONNECT_OBSERVER_AXIS_YAW* is 2.

dAngleDegrees

[in] The angle in degrees to rotate.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
-

-
- [SimConnect_SetObserverRotation](#)
-

SimConnect_SetObserverPosition

The **SimConnect_SetObserverPosition** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverPosition(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_LATLONALT Position,  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

Position

[in] The xyz position to place the observer. In meters.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_MoveObserver](#)
-

SimConnect_SetObserverRotation

The **SimConnect_SetObserverRotation** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverRotation(
    HANDLE hSimConnect,
    const char* szName,
    SIMCONNECT_DATA_PBH RotationAngles,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

RotationAngles

[in] The pitch, bank, and heading for the observer.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Noe.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
-

SimConnect_SetObserverLookAt

The **SimConnect_SetObserverLookAt** function creates an observer using **SIMCONNECT_DATA_OBSERVER**.

Syntax

```
HRESULT SimConnect_SetObserverLookAt(
    HANDLE hSimConnect,
    const char* szName,
    SIMCONNECT_DATA_LATLONALT TargetPosition,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

TargetPosition

[in] The location that the observer should be oriented towards.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
--------------	-------------

S_OK	The function succeeded.
-------------	-------------------------

E_FAIL	The function failed.
---------------	----------------------

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
-

SimConnect_SetObserverLookAtEx

The **SimConnect_SetObserverLookAtEx** function orients the observer specified to look at the round world corrected position specified. This results in a more accurate orientation at larger distances (greater than 2 miles) when compared to **SimConnect_SetObserverLookAt**.

Syntax

```
HRESULT SimConnect_SetObserverLookAtEx(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_LATLONALT TargetPosition,  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer

dialogs of *Prepar3D*.

TargetPosition

[in] The location that the observer should be oriented towards.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
-

SimConnect_ObserverTrackLocationOn

The **SimConnect_ObserverTrackLocationOn** function is used to have an observer track a world position (round world corrected) using an above ground level altitude and accounting for changes in elevation.

Syntax

```
HRESULT SimConnect_ObserverTrackLocationOn(  
    HANDLE hSimConnect,  
    const char* szName,  
    SIMCONNECT_DATA_LATLONALT TargetPosition,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

TargetPosition

[in] The location that the observer should be oriented towards. Altitude is AGL. Units are radians/meters.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
 - [SimConnect_ObserverTrackLocationOff](#)
-

SimConnect_ObserverTrackLocationOff

The **SimConnect_ObserverTrackLocationOff** function is used to disable continuous tracking of the previously specified world position.

Syntax

```
HRESULT SimConnect_ObserverTrackLocationOff(  
    HANDLE hSimConnect,  
    const char* szName,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
--------------	-------------

S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
 - [SimConnect_ObserverTrackLocationOn](#)
-

SimConnect_ObserverAttachToEntityOn

The **SimConnect_ObserverAttachToEntityOn** function is used to attach an observer to the specified entity at the specified offset from the entity.

Syntax

```
HRESULT SimConnect_ObserverAttachToEntityOn(
    HANDLE hSimConnect,
    const char* szName,
    DWORD dwObjectID,
    SIMCONNECT_DATA_XYZ relativeXYZ,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

dwObjectID

[in] The ID of the object to attach to.

relativeXYZ

[in] The offset that the observer should be from the attached entity.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
 - [SimConnect_ObserverAttachToEntityOff](#)
-

SimConnect_ObserverAttachToEntityOff

The **SimConnect_ObserverAttachToEntityOff** function is used to detach an observer from its attached entity.

Syntax

```
HRESULT SimConnect_ObserverAttachToEntityOff(  
    HANDLE hSimConnect,  
    const char* szName,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

-
- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverRotation](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
 - [SimConnect_ObserverAttachToEntityOn](#)
-

SimConnect_SetObserverFieldOfView

The **SimConnect_SetObserverFieldOfView** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverFieldOfView(  
    HANDLE hSimConnect,  
    const char* szName,  
    float fHorizontal,  
    float fVertical,  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

fHorizontal

[in] The horizontal field of view for the observer. In degrees.

fVertical

[in] The vertical field of view for the observer. In degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

-
- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
-

-
- [SimConnect_SetObserverFieldOfView](#)
 - [SimConnect_SetObserverStepSize](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
 - [SimConnect_SetObserverZoomLevels](#)
-

SimConnect_SetObserverStepSize

The **SimConnect_SetObserverStepSize** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverStepSize(  
    HANDLE hSimConnect,  
    const char* szName,  
    float fLinearStep,  
    float fAngularStep,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

fLinearStep

[in] The linear step for the observer view. In meters.

fAngularStep

[in] The angular step for the observer view. In degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_RotateObserver](#)
 - [SimConnect_SetObserverRotation](#)
-

-
- [SimConnect_SetObserverLookAt](#)
 - [SimConnect_SetObserverLookAtEx](#)
 - [SimConnect_ObserverTrackLocationOn](#)
 - [SimConnect_ObserverTrackLocationOff](#)
 - [SimConnect_SetObserverFieldOfView](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
-

SimConnect_SetObserverFocalLength

The **SimConnect_SetObserverFocalLength** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverFocalLength(  
    HANDLE hSimConnect,  
    const char* szName,  
    float fFocalLength,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

fFocalLength

[in] The focal length for the observer. In meters.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_SetObserverFieldOfView](#)
 - [SimConnect_SetObserverStepSize](#)
 - [SimConnect_SetObserverFocusFixed](#)
-

-
- [SimConnect_SetObserverZoomLevels](#)
-

SimConnect_SetObserverFocusFixed

The **SimConnect_SetObserverFocusFixed** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverFocusFixed(  
    HANDLE hSimConnect,  
    const char* szName,  
    BOOL bFocusFixed,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

bFocusFixed

[in] Specifies the observer focal point. If **false**, the observer automatically shifts focal point to the world (terrain) as the observer is manipulated. If **true**, the observer locks the focus to a fixed distance relative to the observer's position.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_SetObserverFieldOfView](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverZoomLevels](#)
-

SimConnect_SetObserverRegime

The **SimConnect_SetObserverRegime** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverRegime(  
    HANDLE hSimConnect,  
    const char* szName,  
    DWORD eRegime,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

eRegime

[in] The restrictions of the observer. Tellurian (earth-based) is 0, Terrestrial (land-based) is 1, and Ghost (unimpeded) is 2.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
- [SimConnect_RequestObserverData](#)
- [SimConnect_MoveObserver](#)
- [SimConnect_SetObserverPosition](#)

SimConnect_SetObserverZoomLevels

The **SimConnect_SetObserverZoomLevels** function creates an observer using SIMCONNECT_DATA_OBSERVER.

Syntax

```
HRESULT SimConnect_SetObserverZoomLevels(  
    HANDLE hSimConnect,
```

```
const char* szName,
DWORD dwNumLevels,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

dwNumLevels

[in] The number of zoom levels to being using.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_SetObserverFieldOfView](#)
 - [SimConnect_SetObserverFocalLength](#)
 - [SimConnect_SetObserverFocusFixed](#)
-

SimConnect_ObserverTrackEntityOn

The **SimConnect_ObserverTrackEntityOn** function sets the observer to continuously track the specified entity.

Syntax

```
HRESULT SimConnect_ObserverTrackEntityOn(
HANDLE hSimConnect,
const char* szName,
DWORD dwObjectID,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that will track the specified object.

dwObjectID

[in] The ID of the object to track.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_ObserverTrackEntityOff](#)
-

SimConnect_ObserverTrackEntityOff

The **SimConnect_ObserverTrackEntityOff** function turns off continuous tracking of the previously specified entity.

Syntax

```
HRESULT SimConnect_ObserverTrackEntityOff(
```

```
    HANDLE hSimConnect,
```

```
    const char* szName,
```

```
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that will no longer track the specified object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
 - [SimConnect_RequestObserverData](#)
 - [SimConnect_ObserverTrackEntityOn](#)
-

SimConnect_SetObserverSceneryOrigin

The **SimConnect_SetObserverSceneryOrigin** function sets scenery origin mode which is used for loading scenery and traffic.

Syntax

```
HRESULT SimConnect_SetObserverSceneryOrigin(
    HANDLE hSimConnect,
    const char* szName,
    DWORD eSceneOrigin,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szName

[in] The name of the observer view that should be updated. This name appears on the observer dialogs of *Prepar3D*.

eSceneOrigin

[in] The observers scenery origin mode. Self (own origin) is 0, and Target (focal point) is 1.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Setting scenery origin to Target can impact performance and memory usage because content will page

in relative to the target which can move rapidly in some modes and may be far away for other cameras used by the sim. If the observer is in world focus the target is determined by projecting the view direction to a point on the ground (even when looking at a simulated object). .

See Also

- [SimConnect_CreateObserver](#)
- [SimConnect_RequestObserverData](#)
- [SimConnect_MoveObserver](#)
- [SimConnect_SetObserverPosition](#)

[- top -](#)

World Functions

Overview

To view a list of all world SimConnect functions, see the [World Functions](#) table.

SimConnect_RequestShadowFlags

The **SimConnect_RequestShadowFlags** function is used to request the current shadow flag settings.

Syntax

```
HRESULT SimConnect_RequestShadowFlags(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Scenery Complexity and Shadow Flags](#)

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_SHADOW_FLAGS](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestSceneryComplexity](#)
-

SimConnect_RequestTrafficSettings

The **SimConnect_RequestTrafficSettings** function is used to request the current traffic settings.

Syntax

```
HRESULT SimConnect_RequestTrafficSettings(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples None

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_TRAFFIC_SETTINGS](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetTrafficSettings](#)
-

SimConnect_SetTrafficSettings

The **SimConnect_SetTrafficSettings** function is used to set the current traffic settings.

Syntax

```
HRESULT SimConnect_SetTrafficSettings(  
    HANDLE hSimConnect,  
    UINT uAirlineDensity,  
    UINT uGADensity,  
    UINT uRoadTrafficDensity,  
    UINT uShipsAndFerriesDensity,  
    UINT uLeisureBoatDensity,  
    SIMCONNECT_DYNAMIC_FREQUENCY eAirportVehicleDensity,  
    BOOL bIFROnly  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

uAirlineDensity

[in] The value that airline traffic density should be set to from 0 to 100.

uGADensity

[in] The value that general aviation traffic density should be set to from 0 to 100.

uRoadTrafficDensity

[in] The value that road traffic density should be set to from 0 to 100.

uShipsAndFerries

[in] The value that ship and ferry traffic density should be set to from 0 to 100.

uLeisureBoatDensity

[in] The value that leisure boat traffic density should be set to from 0 to 100.

eAirportVehicleDensity

[in] The value that airport vehicle traffic density should be set to. See also enumeration

[**SIMCONNECT_DYNAMIC_FREQUENCY**](#).

bIFROnly

[in] Set to TRUE if only IFR traffic should be present, FALSE otherwise.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples

None

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_TRAFFIC_SETTINGS**](#) structure.

See Also

- [**SimConnect API Reference**](#)
 - [**SimConnect_RequestTrafficSettings**](#)
 - [**SIMCONNECT_DYNAMIC_FREQUENCY**](#)
-

SimConnect_RequestSceneryComplexity

The **SimConnect_RequestSceneryComplexity** function is used to request the current scenery complexity setting.

Syntax

```
HRESULT SimConnect_RequestSceneryComplexity(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Scenery Complexity and Shadow Flags](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_SCENERY_COMPLEXITY**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestShadowFlags](#)
-

Facilities Specific Functions

SimConnect_RequestFacilitiesList

The **SimConnect_RequestFacilitiesList** function is used to request a list of all the facilities of a given type currently held in the facilities cache.

Syntax

```
HRESULT SimConnect_RequestFacilitiesList(  
    HANDLE hSimConnect,  
    SIMCONNECT_FACILITY_LIST_TYPE type,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
type
[in] Specifies one member of the [**SIMCONNECT_FACILITY_LIST_TYPE**](#) enumeration type.
RequestID
[in] Specifies the client defined request ID. This will be returned along with the data.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [FacilitiesData](#)

Reference sample [Managed Facilities Request](#)

Remarks

The simulation keeps a facilities cache of all the airports, waypoints, NDB, VOR, and TACAN stations within a certain radius of the user aircraft. This radius varies depending on where the aircraft is in the world, but is at least large enough to encompass the whole of the [reality bubble](#) for airports and waypoints, and can be over 200 miles for VOR and NDB stations. As the user aircraft moves facilities will be added to, and removed from, the cache. However, in the interests of performance, hysteresis is built into the system.

To receive event notifications when a facility is added, use the [SimConnect_SubscribeToFacilities](#) function. When this function is first called, a full list from the cache will be sent, thereafter just the additions will be transmitted. No notification is given when a facility is removed from the cache. Obviously to terminate these notifications use the [SimConnect_UnsubscribeToFacilities](#) function.

When requesting types of facility information, one function call has to be made for each of the four types of data. The data will be returned in one of the four structures:

- [SIMCONNECT_RECV_AIRPORT_LIST](#), which will contain a list of [SIMCONNECT_DATA_FACILITY_AIRPORT](#) structures.
- [SIMCONNECT_RECV_NDB_LIST](#), which will contain a list of [SIMCONNECT_DATA_FACILITY_NDB](#) structures.
- [SIMCONNECT_RECV_VOR_LIST](#), which will contain a list of [SIMCONNECT_DATA_FACILITY_VOR](#) structures.
- [SIMCONNECT_RECV_TACAN_LIST](#), which will contain a list of [SIMCONNECT_DATA_FACILITY_TACAN](#) structures.
- [SIMCONNECT_RECV_WAYPOINT_LIST](#), which will contain a list of [SIMCONNECT_DATA_FACILITY_WAYPOINT](#) structures.

The four list structures inherit the data from the [SIMCONNECT_RECV_FACILITIES_LIST](#) structure. Given that the list of returned facilities could be large, it may be split across several packets, and each packet must be interpreted separately by the client.

See Also

- [SimConnect API Reference](#)
- [SimConnect_SubscribeToFacilities](#)
- [SimConnect_UnsubscribeToFacilities](#)

SimConnect_SubscribeToFacilities

The **SimConnect_SubscribeToFacilities** function is used to request notifications when a facility of a certain type is added to the facilities cache.

Syntax

```
HRESULT SimConnect_SubscribeToFacilities(
    HANDLE hSimConnect,
    SIMCONNECT_FACILITY_LIST_TYPE type,
    SIMCONNECT_DATA_REQUEST_ID RequestID
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

type

[in] Specifies one member of the [SIMCONNECT_FACILITY_LIST_TYPE](#) enumeration type.

RequestID

[in] Specifies the client defined request ID. This will be returned along with the data.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary samples [FacilitiesData](#)
[Managed Facilities Request](#)

Remarks

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect API Reference](#)
 - [SimConnect_UnsubscribeToFacilities](#)
-

SimConnect_UnsubscribeToFacilities

The **SimConnect_UnsubscribeToFacilities** function is used to request that notifications of additions to the facilities cache are not longer sent.

Syntax

```
HRESULT SimConnect_UnsubscribeToFacilities(
    HANDLE hSimConnect,
    SIMCONNECT_FACILITY_LIST_TYPE type
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
type
[in] Specifies one member of the [SIMCONNECT_FACILITY_LIST_TYPE](#) enumeration type.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary samples [FacilitiesData](#)
[Managed Facilities Request](#)

Remarks

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect API Reference](#)
- [SimConnect_SubscribeToFacilities](#)

Ground Info Specific Functions

Overview

The Ground Info API functions are used to request a grid of ground data around a point. There are two API commands available, one provides a one-shot request for data around a specific point, and the other can do one-shot or repeating requests around a specific SimObject. Both versions return a North/South East/West oriented grid. You can have a maximum of 100 points per requested grid, but you can have more than one grid request outstanding.

SimConnect_RequestGroundInfo

The **SimConnect_RequestGroundInfo** function is used to request a grid of altitude points based on a bounding box.

Syntax

```
SimConnect_RequestGroundInfo(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    double minLat,  
    double minLon,  
    double minAlt,
```

```
    double maxLat,
    double maxLon,
    double maxAlt,
    DWORD dwGridWidth,
    DWORD dwGridHeight,
    SIMCONNECT_GROUND_INFO_LATLON_FORMAT eLatLonFormat,
    SIMCONNECT_GROUND_INFO_ALT_FORMAT eAltFormat,
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG eSourceFlags
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which request is receiving a reply. This value should be unique, re-using a *RequestID* will overwrite any previous request using the same ID.

minLat

[in] The minimum Latitude defining the bounding box of the request grid.

minLon

[in] The minimum Longitude defining the bounding box of the request grid.

minAlt

[in] The minimum Altitude defining the bounding box of the request grid.

maxLat

[in] The maximum Latitude defining the bounding box of the request grid.

maxLon

[in] The maximum Longitude defining the bounding box of the request grid.

maxAlt

[in] The maximum Altitude defining the bounding box of the request grid.

dwGridWidth

[in] The number of sample points left/right for the returned grid. The product of dwGridWidth times dwGridHeight must be less than or equal to 100.

dwGridHeight

[in] The number of sample points forward/back for the returned grid. The product of dwGridWidth times dwGridHeight must be less than or equal to 100.

eLatLonFormat

[in] A member of the [SIMCONNECT_GROUND_INFO_LATLON_FORMAT](#) enumeration specifying the format used for Latitude or Longitude values.

eAltFormat

[in] A member of the [SIMCONNECT_GROUND_INFO_ALT_FORMAT](#) enumeration specifying the format used for Altitude values.

eSourceFlags

[in] One or more members of the [SIMCONNECT_GROUND_INFO_SOURCE_FLAG](#) enumeration specifying sources of ground info to use.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
--------------	-------------

S_OK The function succeeded.

E_FAIL The function failed.

See Also

- [SimConnect_RequestGroundInfoOnSimObject](#)
 - [SIMCONNECT_GROUND_INFO_LATLON_FORMAT](#)
 - [SIMCONNECT_GROUND_INFO_ALT_FORMAT](#)
 - [SIMCONNECT_GROUND_INFO_SOURCE_FLAG](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestGroundInfoOnSimObject

Syntax

```
HRESULT SimConnect_RequestGroundInfoOnSimObject(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    DWORD ObjectID,  
    double offsetLat,  
    double offsetLon,  
    double offsetAlt,  
    DWORD dwGridWidth,  
    DWORD dwGridHeight,  
    SIMCONNECT_GROUND_INFO_LATLON_FORMAT eLatLonFormat,  
    SIMCONNECT_GROUND_INFO_ALT_FORMAT eAltFormat,  
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG eSourceFlags  
    SIMCONNECT_PERIOD Period,  
    SIMCONNECT_DATA_REQUEST_FLAG Flags = 0,  
    DWORD origin = 0,  
    DWORD interval = 0,  
    DWORD limit = 0  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which request is receiving a reply. This value should be unique, re-using a *RequestID* will overwrite any previous request using the same ID.

ObjectID

[in] Specifies the ID of the *Prepar3D* object that the ground info grid should be based on. This ID can be **SIMCONNECT_OBJECT_ID_USER** (to specify the user's aircraft) or obtained from a [SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE](#) structure after a call to [SimConnect_RequestDataOnSimObjectType](#).

offsetLat

[in] Latitude offset from SimObject center point to North/South edges of bounding box.

offsetLon

[in] Longitude offset from SimObject center point to East/West edges of bounding box.

offsetAlt

[in] Altitude offset from SimObject center point to Top/Bottom edges of bounding box.

dwGridWidth

[in] The number of sample points East/West for the returned grid. The product of dwGridWidth times dwGridHeight must be less than or equal to 100.

dwGridHeight

[in] The number of sample points North/South for the returned grid. The product of dwGridWidth times dwGridHeight must be less than or equal to 100.

eLatLonFormat

[in] A member of the [SIMCONNECT_GROUND_INFO_LATLON_FORMAT](#) enumeration specifying the format used for Latitude or Longitude values.

eAltFormat

[in] A member of the [SIMCONNECT_GROUND_INFO_ALT_FORMAT](#) enumeration specifying the format used for Altitude values.

eSourceFlags

[in] One or more members of the [SIMCONNECT_GROUND_INFO_SOURCE_FLAG](#) enumeration specifying sources of ground info to use.

Period

[in] One member of the [SIMCONNECT_PERIOD](#) enumeration type, specifying how often the data is to be sent by the server and received by the client.

Flags

[in, optional] A DWORD containing one or more of the following values:

Flag value	Description
0	The default, data will be sent strictly according to the defined period.
SIMCONNECT_DATA_REQUEST_FLAG_CHANGED	Data will only be sent to the client when the SimObject has moved.
SIMCONNECT_DATA_REQUEST_FLAG_BLOCK	Requested data will be sent using a blocking callback. See SimConnect_RequestSynchronousBlock for more information.

origin

[in, optional] The number of *Period* events that should elapse before transmission of the data begins. The default is zero, which means transmissions will start immediately.

interval

[in, optional] The number of *Period* events that should elapse between transmissions of the data. The default is zero, which means the data is transmitted every *Period*.

limit

[in, optional] The number of times the data should be transmitted before this communication is ended. The default is zero, which means the data should be transmitted endlessly.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
--------------	-------------

S_OK The function succeeded.

E_FAIL The function failed.

See Also

- [SimConnect_RequestGroundInfo](#)
- [SIMCONNECT_GROUND_INFO_LATLON_FORMAT](#)
- [SIMCONNECT_GROUND_INFO_ALT_FORMAT](#)
- [SIMCONNECT_GROUND_INFO_SOURCE_FLAG](#)
- [SIMCONNECT_PERIOD](#)
- [SimConnect API Reference](#)

Weather Specific Functions

[SimConnect_WeatherCreateStation](#)

The **SimConnect_WeatherCreateStation** function creates a weather station at the given ICAO location.

Syntax

```
HRESULT SimConnect_WeatherCreateStation(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    const char* szICAO,  
    const char* szName,  
    float lat,  
    float lon,  
    float alt  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

szICAO

[in] Specifies the ICAO string. This can be an existing airport ICAO string, as long as the airport does not already have a weather station, or it can be a unique new ICAO code to be used just for the purposes of this weather station.

szName

[in] Specifies a descriptive name for the new weather station. This name will appear on the weather map in the weather dialogs of *Prepar3D*.

lat

[in] Specifies the latitude of the station in degrees. The latitude, longitude and altitude parameters should be set to 0 if the ICAO code supplied is from an existing airport.

lon

[in] Specifies the longitude of the station in degrees.

alt

[in] Specifies the altitude of the station in feet, above ground level.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value Description

S_OK The function succeeded.

E_FAIL The function failed.

Remarks

If an attempt is made to create a weather station at an airport that already has one, the error **SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_CREATE_STATION** will be returned.

Once a weather station has been successfully created, its weather output can be set with a call to [SimConnect_WeatherSetObservation](#), and retrieved with a call to [SimConnect_WeatherRequestObservationAtStation](#).

See Also

- [SimConnect_WeatherRemoveStation](#)

-
- [SimConnect API Reference](#)
-

SimConnect_WeatherCreateThermal

The **SimConnect_WeatherCreateThermal** function is used to create a thermal at a specific location.

Syntax

```
HRESULT SimConnect_WeatherCreateThermal(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    float lat,  
    float lon,  
    float alt,  
    float radius,  
    float height,  
    float coreRate = 3.0f,  
    float coreTurbulence = 0.05f,  
    float sinkRate = 3.0f,  
    float sinkTurbulence = 0.2f,  
    float coreSize = 0.4f,  
    float coreTransitionSize = 0.1f,  
    float sinkLayerSize = 0.4f,  
    float sinkTransitionSize = 0.1f  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

lat

[in] Specifies the latitude of the thermal in degrees.

lon

[in] Specifies the longitude of the thermal in degrees.

alt

[in] Specifies the altitude of the thermal in feet, above ground level.

radius

[in] Specifies the radius of the thermal, in meters. The maximum radius of a thermal is 100Km.

height

[in] Specifies the height of the thermal, in meters.

coreRate

[in, optional] Specifies the lift value, in meters per second, within the Core layer. A positive value will provide an updraft, a negative value a downdraft. The maximum rate is 1000 meters/second. Refer to the diagram in the Remarks section.

coreTurbulence

[in, optional] Specifies a variation in meters per second that is applied to the *coreRate*. For example, if a value of 1.5 is entered, and the core rate is 5 m/s, the actual core rate applied will be randomly varying between 3.5 m/s and 6.5 m/s.

sinkRate

[in, optional] Specifies the lift value, in meters per second, within the Sink layer. A positive value will provide an updraft, a negative value a downdraft. The maximum rate is 1000 meters/second. Refer to the diagram in the Remarks section.

coreTurbulence

[in, optional] Specifies a variation in meters per second that is applied to the *sinkRate*. For example, if a value of 1.5 is entered, and the sink rate is 5 m/s, the actual sink rate applied will be

randomly varying between 3.5 m/s and 6.5 m/s.

coreSize
[in, optional] Specifies the radius in meters of the Core of the thermal.

coreTransitionSize
[in, optional] Specifies the width in meters of the transition layer between the Core and the Sink of the thermal. Half of the width of this transition will be outside the Core, and half within.

sinkLayerSize
[in, optional] Specifies the radius in meters of the Sink of the thermal.

sinkTransitionSize
[in, optional] Specifies the width in meters of the transition layer between the Sink and the atmosphere outside of the thermal. Half of the width of this transition will be outside the radius of the Sink layer, and half within.

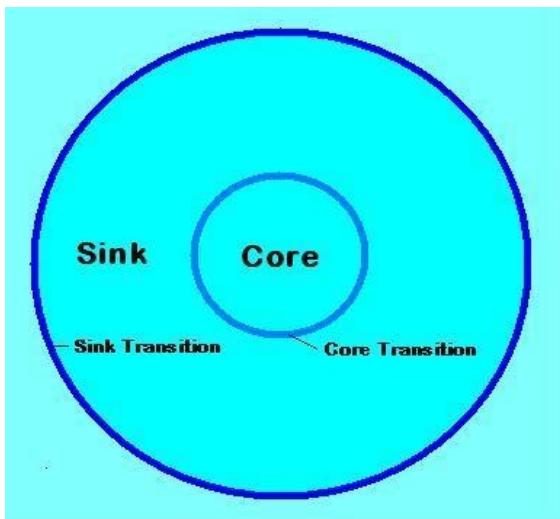
Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

There is no limit to the number of thermals that can be created. Within the simulator a thermal is defined as a cylinder with a Core layer and a Sink layer:



Refer also to the [Weather Systems](#) documentation.

See Also

- [SimConnect API Reference](#)
- [SimConnect_WeatherRemoveThermal](#)
- [SimConnect_WeatherRequestInterpolatedObservation](#)
- [SimConnect_WeatherRequestObservationAtStation](#)
- [SimConnect_WeatherRequestObservationAtNearestStation](#)
- [SimConnect_WeatherSetObservation](#)

SimConnect_WeatherRemoveStation

The **SimConnect_WeatherRemoveStation** function requests that the weather station identified by

the given ICAO string is removed.

Syntax

```
HRESULT SimConnect_WeatherRemoveStation(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    const char* szICAO  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

szICAO

[in] Specifies the ICAO string of the station to remove. The station must be one that was created by the same SimConnect client.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

If a call is made to remove a weather station created by another client, or an exiting one within the simulation, the error

SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_REMOVE_STATION will be returned.

If the client is closed, then all weather stations created by the client will be removed.

See Also

- [SimConnect_WeatherCreateStation](#)
- [SimConnect API Reference](#)

SimConnect_WeatherRemoveThermal

The **SimConnect_WeatherRemoveThermal** function removes a thermal.

Syntax

```
HRESULT SimConnect_WeatherRemoveThermal(  
    HANDLE hSimConnect,  
    SIMCONNECT_OBJECT_ID ObjectID  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
ObjectID
[in] Specifies the object ID of the thermal to be removed.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

A client application can only remove thermals that it created, and not thermals created by other clients or by the simulation itself. If the client is closed, then all thermals created by the client will be removed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_WeatherCreateThermal](#)
 - [SimConnect_WeatherRequestInterpolatedObservation](#)
 - [SimConnect_WeatherRequestObservationAtStation](#)
 - [SimConnect_WeatherRequestObservationAtNearestStation](#)
 - [SimConnect_WeatherSetObservation](#)
-

SimConnect_WeatherRequestCloudState

The **SimConnect_WeatherRequestCloudState** function requests cloud density information on a given area.

Syntax

```
HRESULT SimConnect_WeatherRequestCloudState(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    float minLat,
    float minLon,
    float minAlt,
    float maxLat,
    float maxLon,
    float maxAlt,
    DWORD dwFlags = 0
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] Specifies the client-defined request ID.
minLat
[in] Specifies the minimum latitude of the required area. This should simply be the lower of the two latitude numbers.

minLon

[in] Specifies the minimum longitude of the required area. This should simply be the lower of the two longitude numbers.

minAlt

[in] Specifies the minimum altitude of the required area, in feet.

maxLat

[in] Specifies the maximum latitude of the required area.

maxLon

[in] Specifies the maximum longitude of the required area.

maxAlt

[in] Specifies the maximum altitude of the required area, in feet.

dwFlags

[in, optional] Double word containing any flags. Reserved for future use.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The main purpose of this function is to enable weather radar.

If the call is successful, the cloud state information will be returned in a **SIMCONNECT_RECV_CLOUD_STATE** structure. This structure will contain a two dimensional array of byte data. The array will be 64 x 64 bytes in size, and each byte will contain a value indicating the cloud layer density for each cell. A value of zero would mean no cloud layers, to a maximum of 255 layers. The area defined in this call is divided into 64 by 64 cells, so the size of each cell will be determined by the values given for the parameters above. Note that the entire World's weather is not simulated all the time, but only a region around the user aircraft, with a radius of approximately 128 kilometers, is modeled at any one time. A request for cloud data outside this region will simply return zeros.

The defined area can cross the Equator or the Greenwich Meridian, but it cannot cross the Poles or the International Date Line.

See Also

- [SimConnect_WeatherSetDynamicUpdateRate](#)
 - [SimConnect_WeatherSetModeCustom](#)
 - [SimConnect_WeatherSetModeGlobal](#)
 - [SimConnect_WeatherSetModeTheme](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherRequestInterpolatedObservation

The **SimConnect_WeatherRequestInterpolatedObservation** function is used to send a request for weather data that is interpolated from the weather at the nearest three weather stations.

Syntax

```
HRESULT SimConnect_WeatherRequestInterpolatedObservation(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
```

```
    float lat,
    float lon,
    float alt
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which data has been received.

lat

[in] Specifies latitude in degrees.

lon

[in] Specifies longitude in degrees.

alt

[in] Specifies altitude in feet above sea level. This differs from most weather data altitudes, which are feet above ground level.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
DWORD dwRequestID = 1000;
hr = SimConnect_WeatherRequestInterpolatedObservation(hSimConnect, dwRequestID, -80.0, 44.0, 10000);
```

Remarks

The weather data will be returned in a [SIMCONNECT_RECV_WEATHER_OBSERVATION](#) structure. A number of errors apply specifically to weather data, see the [SIMCONNECT_RECV_EXCEPTION](#) enumeration.

Interpolated weather data can be used to identify suitable locations for thermals and other local weather systems. The process of using the three nearest stations is not without its drawbacks, as all three stations could be in one direction of the specified point, and not a reasonable spread in different directions. Weather data is returned in [Metar data format](#).

In the case that there is no valid weather data at the specified LLA, the global weather data will be returned instead.

See Also

- [SimConnect_WeatherRequestObservationAtStation](#)
- [SimConnect_WeatherRequestObservationAtNearestStation](#)
- [SimConnect_WeatherSetObservation](#)
- [SimConnect API Reference](#)

SimConnect_WeatherRequestObservationAtNearestStation

The **SimConnect_WeatherRequestObservationAtNearestStation** function is used to send a request for the weather data from the weather station nearest to the specified lat/lon position.

Syntax

```
HRESULT SimConnect_WeatherRequestObservationAtNearestStation(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    float lat,  
    float lon  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which data has been received.

lat

[in] Specifies latitude in degrees.

lon

[in] Specifies longitude in degrees.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
DWORD dwRequestID = 1000;  
hr = SimConnect_WeatherRequestObservationAtNearestStation(hSimConnect,  
dwRequestID, -80.0, 44.0);
```

Working Sample

Primary sample [Weather Station](#)

Remarks

The weather data will be returned in a [**SIMCONNECT_RECV_WEATHER_OBSERVATION**](#) structure. A number of errors apply specifically to weather data, see the [**SIMCONNECT_RECV_EXCEPTION**](#) enumeration. Weather data is returned in [Metar data format](#).

In the case that there is no valid weather data at the nearest station, the global weather data will be

returned instead.

See Also

- [SimConnect_WeatherRequestInterpolatedObservation](#)
 - [SimConnect_WeatherRequestObservationAtStation](#)
 - [SimConnect_SetObservation](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherRequestObservationAtStation

The **SimConnect_WeatherRequestObservationAtStation** function requests the weather data from a weather station identified by its ICAO code.

Syntax

```
HRESULT SimConnect_WeatherRequestObservationAtStation(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    const char* szICAO  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the ID of the client defined request. This is used later by the client to identify which data has been received.

szICAO[4]

[in] Null-terminated string specifying the ICAO identification code of the weather station. Typically this is an airport. Set to **GLOB** to retrieve global weather.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Example

```
DWORD dwRequestID = 1000;  
hr = SimConnect_WeatherRequestObservationAtStation(hSimConnect, dwRequestID,  
    "ICAO");
```

Remarks

The weather data will be returned in a [SIMCONNECT_RECV_WEATHER_OBSERVATION](#) structure. A number of errors apply specifically to weather data, see the [SIMCONNECT_RECV_EXCEPTION](#) enumeration. Weather data is returned in [Metar data format](#).

In the case that there is no valid weather data at the specified ICAO, the global weather data will be returned instead.

See Also

- [SimConnect_WeatherRequestInterpolatedObservation](#)
 - [SimConnect_WeatherRequestObservationAtNearestStation](#)
 - [SimConnect_WeatherSetObservation](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherSetDynamicUpdateRate

The **SimConnect_WeatherSetDynamicUpdateRate** function is used to set the rate at which cloud formations change.

Syntax

```
HRESULT SimConnect_WeatherSetDynamicUpdateRate(
    HANDLE hSimConnect,
    DWORD dwRate
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

dwRate

[in] Double word containing the rate. A value of zero indicates that cloud formations do not change at all. Values between 1 and 5 indicate that cloud formations should change from 1 (the slowest) to 5 (the fastest). These settings match those than can be set through the dialogs of *Prepar3D*.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

None.

See Also

- [SimConnect_WeatherRequestCloudState](#)
 - [SimConnect_WeatherSetModeCustom](#)
 - [SimConnect_WeatherSetModeGlobal](#)
 - [SimConnect_WeatherSetModeTheme](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherSetModeCustom

The **SimConnect_WeatherSetModeCustom** function sets the weather mode to user-defined.

Syntax

```
HRESULT SimConnect_WeatherSetModeCustom(
    HANDLE hSimConnect
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Calling this function sets the weather mode to "User-defined weather" in the Weather dialog of *Prepar3D*, so whatever the user has entered for the weather will be used.

See Also

- [SimConnect_WeatherRequestCloudState](#)
 - [SimConnect_WeatherSetDynamicUpdateRate](#)
 - [SimConnect_WeatherSetModeGlobal](#)
 - [SimConnect_WeatherSetModeTheme](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherSetModeGlobal

The **SimConnect_WeatherSetModeGlobal** function sets the weather mode to global, so the same weather data is used everywhere.

Syntax

```
HRESULT SimConnect_WeatherSetModeGlobal(
    HANDLE hSimConnect
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.

E_FAIL	The function failed.
---------------	----------------------

Remarks

There is not an equivalent setting in the weather dialog of *Prepar3D*.

See Also

- [SimConnect_WeatherRequestCloudState](#)
 - [SimConnect_WeatherSetDynamicUpdateRate](#)
 - [SimConnect_WeatherSetModeCustom](#)
 - [SimConnect_WeatherSetModeTheme](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherSetModeServer

The **SimConnect_WeatherSetModeServer** function is used to switch to a local server for weather observation data. This has been deprecated.

Syntax

```
HRESULT SimConnect_WeatherSetModeServer(  
    HANDLE hSimConnect,  
    DWORD dwPort,  
    DWORD dwSeconds  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

dwPort

[in] Double word containing the port number of the weather server. Set this to zero to reset the weather to normal operation.

dwSeconds

[in] Double word containing the amount of time, in seconds, that should elapse between each update. There is a minimum of 60 seconds.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Deprecated.

See Also

- [SimConnect_WeatherRequestCloudState](#)
 - [SimConnect_WeatherRequestInterpolatedObservation](#)
-

-
- [SimConnect_WeatherRequestObservationAtStation](#)
 - [SimConnect_WeatherRequestObservationAtNearestStation](#)
 - [SimConnect_WeatherSetObservation](#)
 - [SimConnect_WeatherSetDynamicUpdateRate](#)
 - [SimConnect_WeatherSetModeCustom](#)
 - [SimConnect_WeatherSetModeGlobal](#)
 - [SimConnect_WeatherSetModeTheme](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherSetModeTheme

The **SimConnect_WeatherSetModeTheme** function is used to set the weather to a particular theme.

Syntax

```
HRESULT SimConnect_WeatherSetModeTheme(  
    HANDLE hSimConnect,  
    const char* szThemeName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szThemeName

[in] Null-terminated string containing the theme path and filename. The path can be either:

- An absolute path (e.g. C:\Program Files\My Company\My Product\myweathertheme)
- A path relative to the *Lockheed Martin\Prepar3D v5* installation folder (e.g. weather\themes\grayrain)
- A path relative to any of the weather theme folders found in any of the weather.cfg [configuration files](#) (e.g. grayrain)
- An empty string to set clear weather

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Calling this function sets the weather mode to "Weather themes" in the Weather dialog of *Prepar3D*.

There are three files associated with a weather theme, for example: grayrain.wt, grayrain.bmp and grayrain.wtb. The wt file contains the description that will appear in the Current Conditions box in the Weather dialog, the bmp file contains the image that will also appear in the weather dialog, and the wtb file contains data in a propriety format that contains the weather information.

See Also

-
- [SimConnect_WeatherRequestCloudState](#)

-
- [SimConnect_WeatherSetDynamicUpdateRate](#)
 - [SimConnect_WeatherSetModeCustom](#)
 - [SimConnect_WeatherSetModeGlobal](#)
 - [SimConnect API Reference](#)
-

SimConnect_WeatherSetObservation

The **SimConnect_WeatherSetObservation** function is used to set the weather at a specific weather station, identified from within the Metar data string.

Syntax

```
HRESULT SimConnect_WeatherSetObservation(  
    HANDLE hSimConnect,  
    DWORD Seconds,  
    const char* szMETAR  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

Seconds

[in] Specifies the time in seconds that the current weather should merge into the new weather. This delay only applies to clouds in a custom weather setting, not to global or other weather settings (where the change will take place almost immediately).

pszMETAR

[in] Null-terminated string containing the METAR data.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Commonly, a [SimConnect_WeatherSetModeGlobal](#) or [SimConnect_WeatherSetModeCustom](#) should be called after this function. [SimConnect_WeatherSetModeGlobal](#) and [SimConnect_WeatherSetModeCustom](#) allow the Weather Settings UI to be properly set.

When setting specific station observations, it is possible for station observations nearby the specified station to be automatically adjusted. The weather system interpolates and changes weather layers for surrounding stations to create a visually appropriate and logical environment. If exact replication of the METAR data is required, the station and surrounding stations can be set multiple times to force the interpolated values to match the desired METAR data.

A number of errors apply specifically to weather data, see the [SIMCONNECT_RECV_EXCEPTION](#) enumeration.

Metar Data Format

The data format used in the simulation for setting and retrieving weather is a modification to the Metar standard. The standard format is open and is described on a number of websites. The Metar data used in the simulation follows the following format. Note that each section is separated by a space, and that spaces cannot occur within a section (with one exception noted in the table). The sections must be in the correct order, though many are optional and need not be present.

Section	Format	Multiple Entries
	CCCC - a three or four character station ID. For example KORD for O'Hare airport in Chicago. The string GLOB can be used to set global weather data.	
STATION ID (has an extension)	<i>Prepar3D</i> extension: Format is &ANNN	No
	This gives the height of the weather station in meters. This information is returned with a get observation, but when setting this extension must not be present.	
REPORT TYPE	Either METAR or SPECI keywords.	No
AUTO	Optional AUTO keyword, for a fully automated report.	No
COR	Optional COR keyword, for a corrected report.	No
	One of the following four formats (the exact number of characters must be present): HHMM DDHHMM HHMMZ DDHHMMZ	
DATETIME	(all are UTC whether or not the Z is present).	No
	Note that this parameter is ignored when setting weather data. When getting the data, if it was set originally by the user through a SimConnect client or the Weather dialog then the time that it was set to will be returned. If the data was retrieved from a weather server, then the time stamp given to it by the server will be returned.	
NIL	Optional NIL keyword, indicating that no weather report could be made. One of the following two formats: DDDSSSUU (steady) DDDSGGXXUU (gusts)	No
	where: DDD = Direction (0-360 degrees, or VRB for variable) SSS = Speed XX = Gust speed UUU = Units, one of: KMH = Kilometers per hour KT = Knots MPS = Meters per second	
	<i>Prepar3D</i> extension follows the "&" character	
SURFACE WINDS (has an extension)	Format of the extension is &DNNNNNTS where NNNN is the depth (height) in meters. The default is 1000 feet or 305m. T part is turbulence, one of: N - None (default) O or L - Light	No

M - Moderate

H - Heavy

S - Severe

S part is wind shear, one of:

G - Gradual (default)

M - Moderate

S - Steep

I - Instantaneous

The default extension would look like this: **&D305NG**

Normal Metar strings do not contain winds aloft. However, you may specify this entry is an extension. Winds aloft can be entered in one of two formats, but do not use both. This section is not a replacement for setting the surface winds. The first format is slightly more flexible than the second, and should directly follow the SURFACE WINDS, the second format should appear at the end of the metar string.

The altitude of a WINDS ALOFT section specifies the top of the layer in which the given wind data is to apply. The bottom of the layer will be determined by the top of any lower layer, either another WINDS ALOFT entry, or the SURFACE WINDS entry.

Format 1:

WINDS ALOFT
(this section is
an optional
extension)

This format is the same as the surface wind data format above except for the extension portion. Instead of specifying a surface layer depth, using **&DNNNNTS**, it specifies the altitude above the referenced weather station's altitude in meters. The extension format is **&ANNNNTS**. Yes

Format 2:

@@@ A T D S | A T D S |

where

A is altitude in 100s of feet (-1500 to 100000)

T is temperature in Celsius (-250 to 100)

D is direction in degrees (0 to 360)

S is speed in knots (0 to 400)

Note that there can be more than one WINDS ALOFT entry, typically giving different wind strengths and the other data, for a range of altitudes.

XXXYYY

MIN MAX WIND DIR XXX - start of heading range, in degrees
YYY - end of heading range, in degrees No

This entry adds a variance in wind direction to the surface wind.

Optional **CAVOK** keyword (meaning Ceiling and Visibility OK). It indicates that no clouds exist below 5,000 feet or below the highest minimum sector altitude, whichever is greater, and no cumulonimbus are present. Also the visibility is 10 kilometers or more and, no precipitation, thunderstorms, sandstorm, duststorm, shallow fog, or low drifting dust, sand or snow is occurring.

Like winds aloft the visibility group can be repeated to describe multiple visibility layers. There is also an extension to the format to allow layer base and depth to be specified.

Use one of the following formats:

Statute Miles:

M1/4SM or **<1/4SM** (visibility is less than a quarter of a statute mile)

ISM (visibility is measured in an integer number of statute miles)

N/DSM (visibility is measured as a fraction of a statute mile)

I N/DSM (visibility is measured in a whole part and fraction of a statute mile)

where

I = Integer part

N = Numerator

D = Denominator

VISIBILITY (has an extension) The minimum fraction that can be entered is 1/8. Note that this is the only exception where a space can occur (between the I and the N) within a section. Yes

Kilometers: NNKM

Meters: NNNND

where

D is directional variation, one of:

NDV - no directional variation

NE, NW, SE, SW, N, S, E, W - compass point

M meters - same as **NDV**

Prepar3D extension:

Format is **&BXXXX&DYYYY**

XXXX - base of visibility layer in meters

YYYY - depth of visibility layer in meters

Use one of the following two formats:

RDD/VVVV/FT

RDD/XXXXVYYYYFT

RUNWAY VISUAL RANGE

where

DD = Runway ID (1-6 characters)

Yes

VVVV = Visual range in feet. May have **P** (above maximum) or **M** (below minimum) prefix (following the slash).

XXXX = Varying minimum range in feet.

YYYY = Varying maximum range in feet.

IDDPP

where:

I - Intensity or vicinity flag: - , + or **VC**. - means light, + means severe, and VC means vicinity. The + and - can be combined with VC. Leave out a + or - to mean moderate.

For example: **+VCTSRA** means severe thunderstorm with rain in the vicinity.

DD - Descriptor, one of:

MI shallow

PR partial

DC patches

DR low drifting

BL blowing

SH shower

TS thunderstorm

FZ freezing

PP - Phenomena, one of:

PRESENT CONDITIONS **DZ** drizzle **Yes**
 RA rain

SN snow

SG snow grains

IC ice crystals

PE ice pellets

GR hail

GS small hail/snow pellets

UP unknown

BR mist

FG fog

FU smoke

VA volcanic ash

DU dust

SA sand

HZ haze

PY spray

PO dust whirls

SQ squalls

FC funnel cloud/tornado/waterspout

SS sandstorm

DS duststorm

One of:

FEW///

SCT///

PARTIAL OBSCURATION **BKN///** **No**
 FEW000
 SCT000
 BKN000

These mean few, scattered or broken clouds are obscuring the view.

Note cloud heights are coded: If NNN is 999 the level is 100,000 feet, otherwise it is 100 x NNN in feet.

CCCN

Where NNN is the coded height, and CCC is one of::

CLR or **SKC** - sky clear

FEW - few clouds

SCT - scattered clouds

BKN - broken clouds

OVC - overcast

NTT - N/8ths cloud coverage of type TT, which is one of:

CI Cirrus

CS Cirro-stratus (maps to **CI**)

CC Cirro-cumulus (maps to **CI**)

AS Alto-stratus (maps to **ST**)

AC Alto-cumulus (maps to **CU**)

SC Strato-cumulus (maps to **CU**)

NS Nimbo-stratus (maps to **ST**)

ST Stratus

SKY CONDITIONS (has an extension)	CU Cumulus CB Cumulo-nimbus Note that not all of these cloud types are supported, so a number are mapped to those which are. This does mean that a write followed by a read of Metar data might not give identical strings.	No
	<i>Prepar3D extension:</i>	
	&TT000FTPQBBBI	
	where: TT - Cloud type, one of: the list above (for example, CI or CB). If this entry is different from the NNN entry above, this entry will take priority. 000 - Unused. F - Top of cloud, one of: F (flat), R (round), A (anvil) T - Turbulence, one of: N - None (default), O - Light, L - Light, M - Moderate, H - Heavy, S - Severe P - Precipitation, one of: V (very light), L (light), M (moderate), H (heavy) D (dense) Q - Type of precipitation, one of: N (none), R (rain), F (freezing rain), H (hail), S (snow) BBB - Coded base height, the precipitation ends at this height, set to 0 for it to land on the ground I - icing rate, one of: N (none), T (trace), L (light), M (moderate), S (severe)	
	TT/DD	
TEMPERATURE (has an extension)	TT - temperature in Celsius DD - dewpoint in Celsius Negative values should be preceded by a minus sign.	Yes
	<i>Prepar3D extension:</i>	
	&ANNNNN - altitude of the temperatures in meters.	
	One of:	
ALTIMETER	ANNNN - altimeter in degrees of mercury (for example, A2992) QNNNN - altimeter in millibars	No
@@@	If these three characters are entered in the string, there is a WINDS ALOFT section following the metar string.	

Examples of Metar strings

	Metar String	Description
KSEA 030405Z 27007KT 15SM SKC 17/13 A2992		KSEA = Station Identifier (Sea-Tac airport) 030405Z = Time (ddhhmmZ) 27007KT = Winds (270 degrees at 7 knots) 15SM = Visibility (15 statute miles) SKC = Clear skies 17/13 = Temperature

KSEA COR 030405Z 27015KT 7SM +TSRA BKN055CB 30/17 A2974	/Dewpoint in Celsius A2992 = Altimeter setting (29.92 in Hg) Corrected report, same time as above, with 15 knot winds, 7 miles of visibility, severe thunderstorm and rain, broken cumulo-nimbus clouds at 5500 feet.
KSEA&A131 000000Z 00000KT&D985NG 100KM&B-581&D3048 2CU053&CU000FNMN-19N 15/05 Q1013 @@@ 65 15 270 20 196 15 270 25	Fair weather at Sea-Tac airport. Two eights cumulus clouds at 5300 feet, 20 knot West winds at 6500 feet, 25 knot West winds at 19600 feet.

See Also

- [SimConnect_WeatherRequestInterpolatedObservation](#)
- [SimConnect_WeatherRequestObservationAtStation](#)
- [SimConnect_WeatherRequestObservationAtNearestStation](#)
- [SimConnect_WeatherSetModeCustom](#)
- [SimConnect_WeatherSetModeGlobal](#)
- [SimConnect API Reference](#)

Effects Specific Functions

Overview

The Effects APIs are used to create effects. Effects can be placed at a location or attached to a SimObject. These effects can also be timed so they don't last forever.

SimConnect_CreateEffect

The **SimConnect_CreateEffect** function is used to create an effect.

Syntax

```
HRESULT SimConnect_CreateEffect(
    HANDLE hSimConnect,
    const char*(effectName,
    SIMCONNECT_DATA_LATLONALT TargetPosition
    SIMCONNECT_DATA_XYZ offset
    BOOL attachToSimObject
    DWORD dwObjectID
    SIMCONNECT_DATA_REQUEST_ID RequestID
    int effectDuration
);
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
effectName

[in] null-terminated string containing the title of the effect to be created.

Effect Examples

`title=fx_explosion`

`title=fx_flamingDebris`

TargetPosition

[in] SIMCONNECT_DATA_LATLONALT containing the location of the effect.

offset

[in] SIMCONNECT_DATA_XYZ containing the offset from the location or the attached point of the SimObject.

attachToSimObject

[in] BOOL determining whether to use the TargetPosition or to attach to the SimObject.

dwObjectID

[in] ObjectID of an existing SimObject the effect should be attached to.

RequestID

[in] RequestID containing the client defined request ID.

effectDuration

[in, optional] int containing the number of seconds that the effect will last. (default value = -1, for unlimited)

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The *effectId*, which can be used by [SimConnect_RemoveEffect](#), will be returned as a [SIMCONNECT_RECV_ASSIGNED_OBJECT_ID](#) structure as the *dwObjectID* parameter.

See Also

- [SimConnect_RemoveEffect](#)
 - [SimConnect API Reference](#)
-

SimConnect_RemoveEffect

The **SimConnect_RemoveEffect** function is used to remove an effect.

Syntax

```
HRESULT SimConnect_RemoveEffect(  
    HANDLE hSimConnect,  
    DWORD effectId  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

effectId

[in] Double Word containing the identifier of the effect to be removed.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_CreateEffect](#)
- [SimConnect API Reference](#)

[- top -](#)

Scenario Functions

Overview

To view a list of all scenario SimConnect functions, see the [Scenario Functions](#) table.

SimConnect_FlightLoad

The **SimConnect_FlightLoad** function is used to load an existing scenario file.

Syntax

```
HRESULT SimConnect_FlightLoad(  
    HANDLE hSimConnect,  
    const char* szFileName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] Null-terminated string containing the path to the scenario file. The path can either be absolute, or relative to the %USERPROFILE%\Documents\Prepar3D v5 Files folder. Scenario files have the extension .FXML. The extension is not required for .FXML files. To load deprecated .FLT files, the .FLT extension must be specified.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Scenario files can be opened using a text editor.

See Also

- [SimConnect_FlightSave](#)
- [SimConnect_FlightPlanLoad](#)
- [SimConnect API Reference](#)

SimConnect_FlightLoadW

The **SimConnect_FlightLoadW** function is used to load an existing scenario file. This version of the

function supports unicode file paths.

Syntax

```
HRESULT SimConnect_FlightLoadW(  
    HANDLE hSimConnect,  
    const wchar_t* szFileName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] Null-terminated string containing the path to the scenario file. The path can either be absolute, or relative to the %USERPROFILE%\Documents\Prepar3D v5 Files folder. Scenario files have the extension .FXML. The extension is not required for .FXML files. To load deprecated .FLT files, the .FLT extension must be specified.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Scenario files can be opened using a text editor.

See Also

- [SimConnect_FlightSaveW](#)
 - [SimConnect_FlightPlanLoadW](#)
 - [SimConnect API Reference](#)
-

SimConnect_FlightPlanLoad

The **SimConnect_FlightPlanLoad** function is used to load an existing flight plan file.

Syntax

```
HRESULT SimConnect_FlightPlanLoad(  
    HANDLE hSimConnect,  
    const char* szFileName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] Null-terminated string containing the path to the flight plan file. Flight plans have the extension .PLN, but no need to enter an extension here. There is no need to enter the full path to the file (just enter the filename) if the scenario file is in the default *Prepar3D v5 Files* directory. The easiest way to create flight plans is to create them from within *Prepar3D* itself, and then save them off for use by the user or AI controlled aircraft.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Flight plan files can be opened using a text editor.

See Also

- [SimConnect_FlightLoad](#)
 - [SimConnect_FlightSave](#)
 - [SimConnect API Reference](#)
-

SimConnect_FlightPlanLoadW

The **SimConnect_FlightPlanLoadW** function is used to load an existing flight plan file. This version of the function supports unicode file paths.

Syntax

```
HRESULT SimConnect_FlightPlanLoadW(  
    HANDLE hSimConnect,  
    const wchar_t* szFileName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] Null-terminated string containing the path to the flight plan file. Flight plans have the extension .PLN, but no need to enter an extension here. There is no need to enter the full path to the file (just enter the filename) if the scenario file is in the default *Prepar3D v5 Files* directory. The easiest way to create flight plans is to create them from within *Prepar3D* itself, and then save them off for use by the user or AI controlled aircraft.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the

following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Flight plan files can be opened using a text editor.

See Also

- [SimConnect_FlightLoadW](#)
 - [SimConnect_FlightSaveW](#)
 - [SimConnect API Reference](#)
-

SimConnect_FlightSave

The **SimConnect_FlightSave** function is used to save the current state of a scenario to a scenario file.

Syntax

```
HRESULT SimConnect_FlightSave(  
    HANDLE hSimConnect,  
    const char* szFileName,  
    const char* sztitle,  
    const char* szDescription,  
    DWORD Flags  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] Null-terminated string containing the path to the scenario file. The path can either be absolute, or relative to the %USERPROFILE%Documents\Prepar3D v5 Files folder. Scenario files have the extension .FXML. If the extension is not specified, it will still save as an .FXML. If the deprecated .FLT extension is specified, it will be overwritten as an .FXML. It is not possible to save deprecated .FLT files.

sztitle

[in] Null-terminated string containing the title of the scenario file. If this is NULL then the szFileName parameter is used as the title.

szDescription

[in] Null-terminated string containing the text to enter in the Description field of the scenario file.

Flags

[in] Unused.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Scenario files can be opened using a text editor.

See Also

- [SimConnect_FlightLoad](#)
 - [SimConnect_FlightPlanLoad](#)
 - [SimConnect API Reference](#)
-

SimConnect_FlightSaveW

The **SimConnect_FlightSaveW** function is used to save the current state of a scenario to a scenario file. This version of the function supports unicode file paths, titles, and descriptions.

Syntax

```
HRESULT SimConnect_FlightSaveW(
    HANDLE hSimConnect,
    const wchar_t* szFileName,
    const wchar_t* sztitle,
    const wchar_t* szDescription,
    DWORD Flags
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] Null-terminated string containing the path to the scenario file. The path can either be absolute, or relative to the `%USERPROFILE%\Documents\Prepar3D v5 Files` folder. Scenario files have the extension .FXML. If the extension is not specified, it will still save as an .FXML. If the deprecated .FLT extension is specified, it will be overwritten as an .FXML. It is not possible to save deprecated .FLT files.

sztitle

[in] Null-terminated string containing the title of the scenario file. If this is NULL then the `szFileName` parameter is used as the title.

szDescription

[in] Null-terminated string containing the text to enter in the Description field of the scenario file.

Flags

[in] Unused.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

Scenario files can be opened using a text editor.

See Also

- [SimConnect_FlightLoadW](#)
 - [SimConnect_FlightPlanLoadW](#)
 - [SimConnect API Reference](#)
-

Structured Scenario Specific Functions

SimConnect_CompleteCustomMissionAction

The **SimConnect_CompleteCustomMissionAction** function is used to complete the scenario action specified by a GUID.

Syntax

```
HRESULT SimConnect_CompleteCustomMissionAction(
    HANDLE hSimConnect,
    const GUID guidInstanceId
);
```

Parameters

hSimConnect
 [in] Handle to a SimConnect object.
 guidInstanceId
 [in] GUID of the custom action. The GUID should be found in the associated object XML file.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [Mission Action](#)

Remarks

A scenario's objects are specified in an XML file. A custom action is defined within this XML file, and will look similar to the following:

```
<SimMission.CustomAction InstanceId="{ GUID }">
<PayLoadString>Any string goes here! </PayLoadString>
<WaitForCompletion>True</WaitForCompletion>
</SimMission.CustomAction>
```

Custom actions provide a mechanism to add complex processing to the basically data-driven scenario system.

The custom action would typically be triggered from within the object XML file (a trigger referencing the GUID of the custom action), though it could be called from within the SimConnect client with a call to [SimConnect_ExecuteMissionAction](#). It is only necessary to call [SimConnect_CompleteCustomMissionAction](#) if the **WaitForCompletion** value is set to **True**.

If the client calls [SimConnect_ExecuteMissionAction](#) from within the code for a custom action, and it is important that this action completes before any other actions are started (that is, **WaitForCompletion** is **True**) then a second custom action should be defined that calls [SimConnect_CompleteCustomMissionAction](#) after that action is complete, and with the GUID of the first custom action as its parameter. The working sample shows this process.

In order to received notifications that a custom action is to be executed, the SimConnect client should use the [SimConnect_SubscribeToSystemEvent](#) call with the *SystemEventName* parameter set to "**CustomMissionActionExecuted**". This will result in the GUID of the custom action, and the **PayLoadString**, being sent to the client in a [SIMCONNECT_RECV_CUSTOM_ACTION](#) structure.

If a scenario requires additional processing on its completion the SimConnect client should use the [SimConnect_SubscribeToSystemEvent](#) call with the *SystemEventName* parameter set to "**MissionCompleted**".

See Also

- [SimConnect_ExecuteMissionAction](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_ExecuteMissionAction

The **SimConnect_ExecuteMissionAction** function is used to execute the scenario action specified by a GUID.

Syntax

```
HRESULT SimConnect_ExecuteMissionAction(
    HANDLE hSimConnect,
    const GUID guidInstanceId
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

guidInstanceId

[in] GUID of the Mission Action. The GUID should be found in the associated object XML file.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Sample

Primary sample [Mission Action](#)

Remarks

A scenario's objects are specified in an XML file.

In order to use **SimConnect_ExecuteMissionAction**, typically there should be at least one custom action within the object XML file. The custom action will initiate the sending of a notification to the SimConnect client, and the client can then both do some processing of its own and run, by calling **SimConnect_ExecuteMissionAction**, one or more actions (spoken text, for example) that are defined within the XML file.

See the remarks for [SimConnect_CompleteCustomMissionAction](#).

See Also

- [SimConnect_CompleteCustomMissionAction](#)
 - [SIMCONNECT_RECV_CUSTOM_ACTION](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestFlightSegmentCount

The **SimConnect_RequestFlightSegmentCount** function is used to request the number of Flight Segment objects in the active scenario.

Syntax

```
HRESULT SimConnect_RequestFlightSegmentCount(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_MISSION_OBJECT_COUNT**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestGoalCount](#)
 - [SimConnect_RequestMissionObjectiveCount](#)
-

SimConnect_RequestFlightSegmentDataByGUID

The **SimConnect_RequestFlightSegmentDataByGUID** function is used to request information about a Flight Segment object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestFlightSegmentDataByGUID(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    const GUID guidInstanceID,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

guidInstanceID

[in] Specifies the instance ID of the Flight Segment.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.

E_FAIL The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_FLIGHT_SEGMENT**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestFlightSegmentDataByIndex](#)
-

SimConnect_RequestFlightSegmentDataByIndex

The **SimConnect_RequestFlightSegmentDataByIndex** function is used to request information about a Flight Segment object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestFlightSegmentDataByIndex(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    int flightSegmentIndex,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

flightSegmentIndex

[in] Specifies the index of the requested Flight Segment in the scenario.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_FLIGHT_SEGMENT**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestFlightSegmentDataByGUID](#)
-

SimConnect_RequestFlightSegmentRangeData

The **SimConnect_RequestFlightSegmentRangeData** function is used to request information about a specific range of a Flight Segment object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestFlightSegmentRangeData(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    const GUID guidInstanceID,  
    int rangeIndex,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

guidInstanceID

[in] Specifies the instance ID of the Flight Segment.

rangeIndex

[in] Specifies the index of the range within the requested Flight Segment in the scenario.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_PARAMETER_RANGE**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestFlightSegmentDataByGUID](#)
 - [SimConnect_RequestFlightSegmentDataByIndex](#)
-

SimConnect_RequestGoalCount

The **SimConnect_RequestGoalCount** function is used to request the number of Goal/Group Goal objects in the active scenario.

Syntax

```
HRESULT SimConnect_RequestGoalCount(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_MISSION_OBJECT_COUNT**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestFlightSegmentCount](#)
 - [SimConnect_RequestMissionObjectiveCount](#)
-

SimConnect_RequestGoalDataByGUID

The **SimConnect_RequestGoalDataByGUID** function is used to request information about a Goal/Group Goal object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestGoalDataByGUID(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    const GUID guidInstanceID,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

guidInstanceID

[in] Specifies the instance ID of the Goal/Group Goal.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_GOAL**](#) structure.

See Also

- [SimConnect API Reference](#)
- [SimConnect_RequestGoalDataByIndex](#)

SimConnect_RequestGoalDataByIndex

The **SimConnect_RequestGoalDataByIndex** function is used to request information about a Goal/Group Goal object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestGoalDataByIndex(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    int goalIndex,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

goalIndex

[in] Specifies the index of the requested Goal/Group Goal object in the scenario.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table. /p>

Return value	Description
--------------	-------------

S_OK The function succeeded.

E_FAIL The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_GOAL**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestGoalDataByGUID](#)
-

SimConnect_RequestChildGoalDataByIndex

The **SimConnect_RequestChildGoalDataByIndex** function is used to request information about a mission objective or group goal's children.

Syntax

```
HRESULT SimConnect_RequestChildGoalDataByIndex(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    const GUID parentGuidInstanceID,
    int goalIndex,
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

parentGuidInstanceId

[in] Specifies the instance ID of the Mission Objective or Group Goal that will have their children queried.

goalIndex

[in] Specifies the index of the child goal to query.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_GOAL_PAIR**](#) structure.

See Also

- [SimConnect API Reference](#)
-

SimConnect_RequestMissionObjectiveCount

The **SimConnect_RequestMissionObjectiveCount** function is used to request the number of Mission Objective objects in the active scenario.

Syntax

```
HRESULT SimConnect_RequestMissionObjectiveCount(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_MISSION_OBJECT_COUNT**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestFlightSegmentCount](#)
 - [SimConnect_RequestGoalCount](#)
-

SimConnect_RequestMissionObjectiveDataByGUID

The **SimConnect_RequestMissionObjectiveDataByGUID** function is used to request information about a Mission Objective object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestMissionObjectiveDataByGUID(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    const GUID guidInstanceID,
);
```

Parameters

hSimConnect
 [in] Handle to a SimConnect object.
RequestID
 [in] Specifies the client defined request ID.
guidInstanceID
 [in] Specifies the instance ID of the Mission Objective.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_MISSION_OBJECTIVE**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestMissionObjectiveDataByIndex](#)
-

SimConnect_RequestMissionObjectiveDataByIndex

The **SimConnect_RequestMissionObjectiveDataByIndex** function is used to request information about a Mission Objective object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestMissionObjectiveDataByIndex(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID,  
    int missionObjectiveIndex,  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

missionObjectiveIndex

[in] Specifies the index of the requested Mission Objective object in the scenario.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_MISSION_OBJECTIVE**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestMissionObjectiveDataByGUID](#)
-

SimConnect_RequestLandingTriggerCount

The **SimConnect_RequestLandingTriggerCount** function is used to request the number of Area Landing Trigger or Airport Landing Trigger objects in the active scenario.

Syntax

```
HRESULT SimConnect_RequestLandingTriggerCount(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_MISSION_OBJECT_COUNT**](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestLandingTriggerLandingInfoCount](#)
 - [SimConnect_RequestLandingTriggerLandingInfoByIndex](#)
-

SimConnect_RequestLandingTriggerLandingInfoCount

The **SimConnect_RequestLandingTriggerLandingInfoCount** function is used to request information about a landing trigger object's landings in the active scenario.

Syntax

```
HRESULT SimConnect_RequestLandingTriggerLandingInfoCount(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    int landingTriggerIndex
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

landingTriggerIndex

[in] Specifies the index of the requested Landing Trigger object in the scenario.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_LANDING_TRIGGER_INFO](#) structure.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestLandingTriggerCount](#)
 - [SimConnect_RequestLandingTriggerLandingInfoByIndex](#)
-

SimConnect_RequestLandingTriggerLandingInfoByIndex

The **SimConnect_RequestLandingTriggerLandingInfoByIndex** function is used to request information about a specific landing of a Landing Trigger object in the active scenario.

Syntax

```
HRESULT SimConnect_RequestLandingTriggerLandingInfoByIndex(
    HANDLE hSimConnect,
    SIMCONNECT_DATA_REQUEST_ID RequestID,
    const GUID landingTriggerInstanceID,
    int landingIndex
);
```

Parameters

hSimConnect
 [in] Handle to a SimConnect object.
RequestID
 [in] Specifies the client defined request ID.
landingTriggerInstanceID
 [in] Specifies the instance ID of the Landing Trigger.
landingIndex
 [in] Specifies the index of the landing within the requested Landing Trigger in the scenario.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [**SIMCONNECT_RECV_LANDING_INFO**](#) structure.

See Also

- [SimConnect API Reference](#)
- [SimConnect_RequestLandingTriggerCount](#)
- [SimConnect_RequestLandingTriggerLandingInfoCount](#)

SimConnect_ResolveGoal

The **SimConnect_ResolveGoal** function is used to resolve a goal to specified goal state.

```
HRESULT SimConnect_ResolveGoal(  
    HANDLE hSimConnect,  
    const GUID guidInstanceID,  
    SIMCONNECT_GOAL_RESOLUTION goalResolution  
)
```

Parameters

hSimConnect
 [in] Handle to a SimConnect object.
RequestID
 [in] Specifies the client defined request ID.
guidInstanceID
 [in] Specifies the instance ID of the Flight Segment.
goalResolution
 [in] Specifies the state, defined in [**SIMCONNECT_GOAL_RESOLUTION**](#), the goal/group goal should resolve to.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

SimConnect_RequestMobileSceneryInRadius

The **SimConnect_RequestMobileSceneryInRadius** function is used to retrieve mobile scenery objects that are within a specified radius of the user's vehicle. Any request over 20,000 meters will return all mobile scenery objects.

Syntax

```
HRESULT SimConnect_RequestMobileSceneryInRadius(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    DWORD dwRadiusMeters  
);
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

dwRadiusMeters

[in] Specifies the radius (in meters) from the user vehicle from which to obtain mobile scenery.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a

[**SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS**](#) structure.

Working Samples

Primary samples See the [Managed Mission Objects](#) sample.

See Also

-
- [SimConnect_RequestMobileSceneryDataByID](#)
 - [SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS](#)
 - [SIMCONNECT_DATA_MOBILE_SCENERY_INFO](#)
 - [SIMCONNECT_RECV_MOBILE_SCENERY_DATA](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestMobileSceneryDataByID

The **SimConnect_RequestMobileSceneryDataByID** function is used to retrieve data about a specific mobile scenery object using its object ID.

Syntax

```
HRESULT SimConnect_RequestMobileSceneryDataByID(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    SIMCONNECT_OBJECT_ID ObjectID  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] Specifies the client defined request ID.

ObjectID

[in] Specifies the object ID of the mobile scenery object to retrieve data about.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_MOBILE_SCENERY_DATA](#) structure.

Working Samples

Primary samples See the [Managed Mission Objects](#) sample.

See Also

- [SimConnect_RequestMobileSceneryInRadius](#)
 - [SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS](#)
-

-
- [**SIMCONNECT_DATA_MOBILE_SCENERY_INFO**](#)
 - [**SIMCONNECT_RECV_MOBILE_SCENERY_DATA**](#)
 - [**SimConnect API Reference**](#)

- top -

Analysis Functions

Overview

To view a list of all analysis SimConnect functions, see the [Analysis Functions](#) table.

SimConnect_GenerateFlightAnalysisDiagrams

The **SimConnect_GenerateFlightAnalysisDiagrams** function is primarily used to generate diagrams as shown in the flight analysis UI which can be used, for example, to examine ILS landing performance. The following images will be generated in your Prepar3D v5 Files in the My Pictures Library:

- **FlightAnalysis_Plot_Altitude.bmp**: Glide Slope or Distance vs. Altitude Plot (when Glide Slope is not available)
- **FlightAnalysis_Map_ILS.bmp**: Map View showing any ILS localizers

Syntax

```
HRESULT SimConnect_GenerateFlightAnalysisDiagrams(  
    HANDLE hSimConnect  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

The information requested will be returned in a [SIMCONNECT_RECV_EVENT_FLIGHT_ANALYSIS_DIAGRAMS](#) structure.

See Also

- [SimConnect API Reference](#)

Recorder Specific Functions

Overview

The Recorder APIs are used to start and stop the Prepar3D recorder as well as playback specified recordings.

SimConnect_PlaybackRecording

The **SimConnect_PlaybackRecording** function is used to playback a specified recording from a specified start time to a specified end time.

Syntax

```
HRESULT SimConnect_PlaybackRecording(  
    HANDLE hSimConnect,  
    const char * szFileName  
    int bookmarkIndex  
    double endTimeInSeconds  
    BOOL bDisplayPlaybackCompleteDialog = TRUE  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] The file path of the recording to playback.

bookmarkIndex

[in] The bookmark index where playback should begin (0 is the start of the recording, 1 is the first bookmark index).

endTimeInSeconds

[in] The time (in seconds) within the recording where the playback should end (-1 is the end of the recording).

bDisplayPlaybackCompleteDialog

[in] Sets whether or not a dialog prompt will show when playback is complete. This dialog offers the option to continue the scenario where the recording finished or restart the current scenario. Setting this to FALSE will automatically continue the scenario where the recording finished.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

You must specify the full path to the recording in order to play it using **SimConnect_PlaybackRecording**.

See Also

-
- [SimConnect_StartRecorder](#)
 - [SimConnect_StopRecorderAndSaveRecording](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_PlaybackRecordingW

The **SimConnect_PlaybackRecordingW** function is used to playback a specified recording from a specified start time to a specified end time. This version of the function supports unicode file paths.

Syntax

```
HRESULT SimConnect_PlaybackRecordingW(  
    HANDLE hSimConnect,  
    const wchar_t * szFileName  
    int bookmarkIndex  
    double endTimeInSeconds  
    BOOL bDisplayPlaybackCompleteDialog = TRUE  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szFileName

[in] The file path of the recording to playback.

bookmarkIndex

[in] The bookmark index where playback should begin (0 is the start of the recording, 1 is the first bookmark index).

endTimeInSeconds

[in] The time (in seconds) within the recording where the playback should end (-1 is the end of the recording).

bDisplayPlaybackCompleteDialog

[in] Sets whether or not a dialog prompt will show when playback is complete. This dialog offers the option to continue the scenario where the recording finished or restart the current scenario. Setting this to FALSE will automatically continue the scenario where the recording finished.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

You must specify the full path to the recording in order to play it using **SimConnect_PlaybackRecording**.

See Also

-
- [SimConnect_StartRecorder](#)
 - [SimConnect_StopRecorderAndSaveRecording](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_StartRecorder

The **SimConnect_StartRecorder** function is used to trigger the Prepar3D Recorder to start recording.

Syntax

```
HRESULT SimConnect_StartRecorder(  
    HANDLE hSimConnect,  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_PlaybackRecording](#)
 - [SimConnect_StopRecorderAndSaveRecording](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_StopRecorderAndSaveRecording

The **SimConnect_StopRecorderAndSaveRecording** function is used to trigger the Prepar3D Recorder to stop recording and either prompt the user or save the recording with a specified file name.

Syntax

```
HRESULT SimConnect_StopRecorderAndSaveRecording(  
    HANDLE hSimConnect,  
    const char * szTitle  
    const char * szDescription  
    BOOL promptUser  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

szTitle

[in] The title and file name that the recording will be saved with.

szDescription

[in] The description that the recording will be saved with.

promptUser

[in] If true, rather than use the specified title and description, the user is prompted to specify the title and description for the recording.

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

See Also

- [SimConnect_PlaybackRecording](#)
 - [SimConnect_StartRecorder](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestRecordingInfo

The **SimConnect_RequestRecordingInfo** function is used to request information on a given recording file.

Syntax

```
HRESULT SimConnect_RequestRecordingInfo(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    const char * szFileName  
    UINT cbszFileName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] RequestID containing the client defined request ID.

szFileName

[in] The file path of the recording.

cbszFileName

[in] The size of the file path in bytes (not currently used).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

You must specify the full path to the recording.

See Also

- [SimConnect_StartRecorder](#)
 - [SimConnect_StopRecorderAndSaveRecording](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestRecordingInfoW

The **SimConnect_RequestRecordingInfoW** function is used to request information on a given recording file. This version of the function supports unicode file paths.

Syntax

```
HRESULT SimConnect_RequestRecordingInfoW(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    const wchar_t * szFileName  
    UINT cbszFileName  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] RequestID containing the client defined request ID.
szFileName
[in] The file path of the recording.
cbszFileName
[in] The size of the file path in bytes (not currently used).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

You must specify the full path to the recording.

See Also

- [SimConnect_StartRecorder](#)
 - [SimConnect_StopRecorderAndSaveRecording](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestBookmarkInfo

The **SimConnect_RequestBookmarkInfo** function is used to request bookmark information on a given recording file.

Syntax

```
HRESULT SimConnect_RequestBookmarkInfo(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    const char * szFileName  
    UINT cbszFileName  
)
```

Parameters

hSimConnect

[in] Handle to a SimConnect object.

RequestID

[in] RequestID containing the client defined request ID.

szFileName

[in] The file path of the recording.

cbszFileName

[in] The size of the file path in bytes (not currently used).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

You must specify the full path to the recording.

See Also

-
- [SimConnect_StartRecorder](#)
 - [SimConnect_StopRecorderAndSaveRecording](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SimConnect_RequestBookmarkInfoW

The **SimConnect_RequestBookmarkInfoW** function is used to request bookmark information on a given recording file. This version of the function supports unicode file paths.

Syntax

```
HRESULT SimConnect_RequestBookmarkInfoW(  
    HANDLE hSimConnect,  
    SIMCONNECT_DATA_REQUEST_ID RequestID  
    const wchar_t * szFileName  
    UINT cbszFileName  
)
```

Parameters

hSimConnect
[in] Handle to a SimConnect object.
RequestID
[in] RequestID containing the client defined request ID.
szFileName
[in] The file path of the recording.
cbszFileName
[in] The size of the file path in bytes (not currently used).

Return Values

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return value	Description
S_OK	The function succeeded.
E_FAIL	The function failed.

Remarks

You must specify the full path to the recording.

See Also

- [SimConnect_StartRecorder](#)
 - [SimConnect_StopRecorderAndSaveRecording](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

Structures and Enumerations

Overview

To view a list of all general SimConnect structures and enumerations, see the [Structures and Enumerations](#) table.

SIMCONNECT_CLIENT_DATA_PERIOD

The **SIMCONNECT_CLIENT_DATA_PERIOD** enumeration type is used with the [SimConnect_RequestClientData](#) call to specify how often data is to be sent to the client.

Syntax

```
enum SIMCONNECT_CLIENT_DATA_PERIOD{
    SIMCONNECT_CLIENT_DATA_PERIOD_NEVER,
    SIMCONNECT_CLIENT_DATA_PERIOD_ONCE,
    SIMCONNECT_CLIENT_DATA_PERIOD_VISUAL_FRAME,
    SIMCONNECT_CLIENT_DATA_PERIOD_ON_SET,
    SIMCONNECT_CLIENT_DATA_PERIOD_SECOND,
};
```

Members

SIMCONNECT_CLIENT_DATA_PERIOD_NEVER

Specifies that the data is not to be sent.

SIMCONNECT_CLIENT_DATA_PERIOD_ONCE

Specifies that the data should be sent once only. Note that this is not an efficient way of receiving data frequently, use one of the other periods if there is a regular frequency to the data request.

SIMCONNECT_CLIENT_DATA_PERIOD_VISUAL_FRAME

Specifies that the data should be sent every visual (rendered) frame.

SIMCONNECT_CLIENT_DATA_PERIOD_ON_SET

Specifies that the data should be sent whenever it is set.

SIMCONNECT_CLIENT_DATA_PERIOD_SECOND

Specifies that the data should be sent once every second.

Remarks

Although the period definitions are specific, data is always transmitted at the end of a frame, so even if you have specified that data should be sent every second, the data will actually be transmitted at the end of the frame that comes on or after one second has elapsed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetClientData](#)
-

SIMCONNECT_DATA_RACE_RESULT

The **SIMCONNECT_DATA_RACE_RESULT** structure is used to hold multiplayer racing results.

Syntax

```
struct SIMCONNECT_DATA_RACE_RESULT{
```

```
DWORD dwNumberOfRacers;
GUID MissionGUID;
char szPlayerName[MAX_PATH];
char szSessionType[MAX_PATH];
char szAircraft[MAX_PATH];
char szPlayerRole[MAX_PATH];
double fTotalTime;
double fPenaltyTime;
DWORD dwIsDisqualified;
};
```

Members

dwNumberOfRacers

The total number of racers.

MissionGUID

The GUID of the scenario that has been selected by the host.

szPlayerName[MAX_PATH]

Null terminated string containing the name of the player.

szSessionType[MAX_PATH]

Null terminated string containing the type of the multiplayer session, currently one of: "**LAN**" or "**GAMESPY**".

szAircraft[MAX_PATH]

Null terminated string containing the aircraft type, which is the title field from the [Aircraft Configuration File](#).

szPlayerRole[MAX_PATH]

Null terminated string containing the player's role (or name) in the scenario. This string will be filled from the Name property of the Player object in the scenario object file.

fTotalTime

If this structure is a member of a [**SIMCONNECT_RECV_EVENT_RACE_END**](#) structure, then this contains the final race time in seconds, or 0 for DNF (Did Not Finish). If this structure is a member of a [**SIMCONNECT_RECV_EVENT_RACE_LAP**](#) structure, then this contains the lap time in seconds.

fPenaltyTime

If this structure is a member of a [**SIMCONNECT_RECV_EVENT_RACE_END**](#) structure, then this contains the final penalty time in seconds. If this structure is a member of a [**SIMCONNECT_RECV_EVENT_RACE_LAP**](#) structure, then this contains the total penalty time in seconds received so far (not just for this lap).

dwIsDisqualified

A boolean value, 0 indicating the player has not been disqualified, non-zero indicating they have been disqualified.

Remarks

This structure is never sent on its own, but is always a member of either a [**SIMCONNECT_RECV_EVENT_RACE_END**](#) structure or a [**SIMCONNECT_RECV_EVENT_RACE_LAP**](#) structure.

See Also

- [SimConnect API Reference](#)

SIMCONNECT_DATATYPE

The **SIMCONNECT_DATATYPE** enumeration type is used with the [SimConnect>AddToDataDefinition](#) call to specify the data type that the server should use to return the specified data to the client.

Syntax

```
enum SIMCONNECT_DATATYPE
{
    SIMCONNECT_DATATYPE_INVALID,
    SIMCONNECT_DATATYPE_INT32,
    SIMCONNECT_DATATYPE_INT64,
    SIMCONNECT_DATATYPE_FLOAT32,
    SIMCONNECT_DATATYPE_FLOAT64,
    SIMCONNECT_DATATYPE_STRING8,
    SIMCONNECT_DATATYPE_STRING32,
    SIMCONNECT_DATATYPE_STRING64,
    SIMCONNECT_DATATYPE_STRING128,
    SIMCONNECT_DATATYPE_STRING256,
    SIMCONNECT_DATATYPE_STRING260,
    SIMCONNECT_DATATYPE_STRINGV,
    SIMCONNECT_DATATYPE_INITPOSITION,
    SIMCONNECT_DATATYPE_MARKERSTATE,
    SIMCONNECT_DATATYPE_WAYPOINT,
    SIMCONNECT_DATATYPE_LATLONALT,
    SIMCONNECT_DATATYPE_XYZ,
    SIMCONNECT_DATATYPE_PBH,
    SIMCONNECT_DATATYPE_OBSERVER,
    SIMCONNECT_DATATYPE_VIDEO_STREAM_INFO,
    SIMCONNECT_DATATYPE_WSTRING8,
    SIMCONNECT_DATATYPE_WSTRING32,
    SIMCONNECT_DATATYPE_WSTRING64,
    SIMCONNECT_DATATYPE_WSTRING128,
    SIMCONNECT_DATATYPE_WSTRING256,
    SIMCONNECT_DATATYPE_WSTRING260,
    SIMCONNECT_DATATYPE_WSTRINGV,
    SIMCONNECT_DATATYPE_MAX
};
```

Members

SIMCONNECT_DATATYPE_INT32,64

Specifies a 32 bit or 64 bit signed integer.

SIMCONNECT_DATATYPE_FLOAT32,64

Specifies a 32 bit or 64 bit signed floating point number.

SIMCONNECT_DATATYPE_STRING8,32,64,128,256,260

Specifies narrow strings of the given length (8 characters to 260 characters)

SIMCONNECT_DATATYPE_STRINGV

Specifies a variable length narrow string.

SIMCONNECT_DATATYPE_INITPOSITION

Specifies the [SIMCONNECT_DATA_INITPOSITION](#) structure.

SIMCONNECT_DATATYPE_MARKERSTATE

Specifies the [SIMCONNECT_DATA_MARKERSTATE](#) structure.

SIMCONNECT_DATATYPE_WAYPOINT

Specifies the [SIMCONNECT_DATA_WAYPOINT](#) structure.

SIMCONNECT_DATATYPE_LATLONALT

Specifies the [SIMCONNECT_DATA_LATLONALT](#) structure.

SIMCONNECT_DATATYPE_XYZ

Specifies the [SIMCONNECT_DATA_XYZ](#) structure.

SIMCONNECT_DATATYPE_OBSERVER

Specifies the [SIMCONNECT_DATA_OBSERVER](#) structure.

SIMCONNECT_DATATYPE_VIDEO_STREAM_INFO

Specifies the [SIMCONNECT_DATA_VIDEO_STREAM_INFO](#) structure.

SIMCONNECT_DATATYPE_WSTRING8,32,64,128,256,260

Specifies wide strings of the given length (8 characters to 260 characters)

SIMCONNECT_DATATYPE_WSTRINGV

Specifies a variable length wide string.

Working Samples

[Request Data](#)

[Set Data](#)

Primary samples

[Tagged Data](#)

[Throttle Control](#)

Remarks

The three structures in the list of data types can only be used as input (using [SimConnect_SetDataOnSimObject](#)) and not to receive requested data.

See Also

- [SimConnect API Reference](#)

SIMCONNECT_DATA_GROUND_INFO

The **SIMCONNECT_DATA_GROUND_INFO** structure is used to return information on a single ground info point. The [SIMCONNECT_RECV_GROUND_INFO](#) structure contains an array of these, one for each point in the grid array returned by calls to [SimConnect_RequestGroundInfo](#) or [SimConnect_RequestGroundInfoOnSimObject](#).

Syntax

```
struct SIMCONNECT_DATA_GROUND_INFO {
    BOOL bIsValid;
    double fLat;
    double fLon;
    double fAlt;
    double fNormalI;
    double fNormalJ;
    double fNormalK;
    DWORD eSurfaceType;
    DWORD eSurfaceCondition;
    BOOL bIsPlatform;
    BOOL bIsPlatformMoving;
};
```

Members

bIsValid;

Boolean value, true means this data point is valid, false means this data point is invalid.

fLat;

Double floating point value containing the Latitude of this point (units based on value of [SIMCONNECT_RECV_GROUND_INFO.dwFlags](#)).

fLon;

Double floating point value containing the Longitude of this point (units based on value of **SIMCONNECT_RECV_GROUND_INFO.dwFlags**).

fAlt;

Double floating point value containing the Altitude of this point (units based on value of **SIMCONNECT_RECV_GROUND_INFO.dwFlags**).

fNormalI;

Double floating point value containing the I component of the normal at this point.

fNormalJ;

Double floating point value containing the J component of the normal at this point.

fNormalK;

Double floating point value containing the K component of the normal at this point.

eSurfaceType;

Double word containing the type of surface at this point.

eSurfaceCondition;

Double word containing the surface conditions as this point.

bIsPlatform;

Boolean value, true means this point refers to a platform instead of the ground.

bIsPlatformMoving;

Boolean value, true if this point refers to a platform and the platform is moving

Remarks

An array of this structure is part of the [SIMCONNECT_RECV_GROUND_INFO](#) structure. Each element in the array provides info on one ground point.

See the remarks for [SimConect_RequestGroundInfo](#).

See Also

- [SimConnect_RequestGroundInfo](#)
 - [SimConnect_RequestGroundInfoOnSimObject](#)
 - [SIMCONNECT_RECV_GROUND_INFO](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_DATA_FACILITY_AIRPORT

The **SIMCONNECT_DATA_FACILITY_AIRPORT** structure is used to return information on a single airport in the facilities cache.

Syntax

```
struct SIMCONNECT_DATA_FACILITY_AIRPORT{
    char Icao[9];
    double Latitude;
    double Longitude;
    double Altitude;
};
```

Members

Icao[9]

ICAO of the facility.

Latitude

Latitude of the airport in facility.

Longitude

Longitude of the airport in facility.

Altitude

Altitude of the facility in meters.

Remarks

This structure is returned as one element in the [SIMCONNECT_RECV_AIRPORT_LIST](#) structure. Note that this structure is inherited by [SIMCONNECT_DATA_FACILITY_WAYPOINT](#), [SIMCONNECT_DATA_FACILITY_NDB](#), [SIMCONNECT_DATA_FACILITY_VOR](#), and [SIMCONNECT_DATA_FACILITY_TACAN](#), so the latitude, longitude, and altitude will apply to those facilities in that case.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect API Reference](#)
-

SIMCONNECT_DATA_FACILITY_NDB

The **SIMCONNECT_DATA_FACILITY_NDB** structure is used to return information on a single NDB station in the facilities cache.

Syntax

```
struct SIMCONNECT_DATA_FACILITY_NDB : public  
SIMCONNECT_DATA_FACILITY_WAYPOINT{  
    DWORD fFrequency;  
};
```

Members

fFrequency

Frequency of the station in Hz.

Remarks

This structure is returned as one element in the [SIMCONNECT_RECV_NDB_LIST](#) structure. It inherits all the members of the [SIMCONNECT_DATA_FACILITY_WAYPOINT](#) structure.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SIMCONNECT_DATA_FACILITY_AIRPORT](#)
 - [SIMCONNECT_DATA_FACILITY_VOR](#)
 - [SIMCONNECT_DATA_FACILITY_TACAN](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_DATA_FACILITY_TACAN

The **SIMCONNECT_DATA_FACILITY_TACAN** structure is used to return information on a single TACAN station in the facilities cache.

Syntax

```
struct SIMCONNECT_DATA_FACILITY_TACAN : public  
SIMCONNECT_DATA_FACILITY_WAYPOINT{  
    DWORD uChannel;  
    BOOL bXYBandsY;  
};
```

Members

uChannel

The specified channel for this station

bXYBandIsY

True if the operating band is Y instead of X.

Remarks

This structure is returned as one element in the [SIMCONNECT_RECV_TACAN_LIST](#) structure. It inherits all the members of the [SIMCONNECT_DATA_FACILITY_WAYPOINT](#) structure.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SIMCONNECT_DATA_FACILITY_WAYPOINT](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_DATA_FACILITY_VOR

The **SIMCONNECT_DATA_FACILITY_VOR** structure is used to return information on a single VOR station in the facilities cache.

Syntax

```
struct SIMCONNECT_DATA_FACILITY_VOR: public SIMCONNECT_DATA_FACILITY_NDB{
    DWORD Flags;
    float fLocalizer;
    double GlideLat;
    double GlideLon;
    double GlideAlt;
    float fGlideSlopeAngle;
};
```

Members

Flags

Flags indicating whether the other fields are valid or not.

	Flag	Value	Description
SIMCONNECT_RECV_ID_VOR_LIST_HAS_NAV_SIGNAL	SIMCONNECT_RECV_ID_VOR_LIST_HAS_NAV_SIGNAL	0x1	Set if the station has a NAV transmitter, and if so, GlideLat , GlideLon and GlideAlt contain valid data.
SIMCONNECT_RECV_ID_VOR_LIST_HAS_LOCALIZER	SIMCONNECT_RECV_ID_VOR_LIST_HAS_LOCALIZER	0x2	Set if the station transmits an ILS localizer angle, and if so fLocalizer contains valid data.
SIMCONNECT_RECV_ID_VOR_LIST_HAS_GLIDE_SLOPE	SIMCONNECT_RECV_ID_VOR_LIST_HAS_GLIDE_SLOPE	0x4	Set if the station transmits an ILS approach angle, and if so fGlideSlopeAngle contains valid data.
SIMCONNECT_RECV_ID_VOR_LIST_HAS_DME	SIMCONNECT_RECV_ID_VOR_LIST_HAS_DME	0x8	Set if the station transmits a DME signal, and if so the inherited DME fFrequency contains valid data.

fLocalizer

The ILS localizer angle in degrees.

GlideLat

The latitude of the glide slope transmitter in degrees.

GlideLon

The longitude of the glide slope transmitter in degrees.

GlideAlt

The altitude of the glide slope transmitter in degrees.

fGlideSlopeAngle

The ILS approach angle in degrees.

Remarks

This structure is returned as one element in the [**SIMCONNECT_RECV_VOR_LIST**](#) structure. It inherits all the members from [**SIMCONNECT_DATA_FACILITY_NDB**](#).

See the remarks for [**SimConnect_RequestFacilitiesList**](#).

See Also

- [**SIMCONNECT_DATA_FACILITY_AIRPORT**](#)
 - [**SIMCONNECT_DATA_FACILITY_WAYPOINT**](#)
 - [**SimConnect API Reference**](#)
-

SIMCONNECT_DYNAMIC_FREQUENCY

The **SIMCONNECT_DYNAMIC_FREQUENCY** enumeration type is used with the [**SimConnect_RequestTrafficSettings**](#) and [**SimConnect_SetTrafficSettings**](#) functions.

Syntax

```
enum SIMCONNECT_DYNAMIC_FREQUENCY {
    SIMCONNECT_DYNAMIC_FREQUENCY VERY_SPARSE,
    SIMCONNECT_DYNAMIC_FREQUENCY SPARSE,
    SIMCONNECT_DYNAMIC_FREQUENCY NORMAL,
    SIMCONNECT_DYNAMIC_FREQUENCY DENSE,
    SIMCONNECT_DYNAMIC_FREQUENCY VERY_DENSE,
    SIMCONNECT_DYNAMIC_FREQUENCY EXTREMELY_DENSE,
};
```

Members

SIMCONNECT_DYNAMIC_FREQUENCY VERY_SPARSE

The current dynamic scenery frequency setting is set to Very Sparse.

SIMCONNECT_DYNAMIC_FREQUENCY SPARSE

The current dynamic scenery frequency setting is set to Sparse.

SIMCONNECT_DYNAMIC_FREQUENCY NORMAL

The current dynamic scenery frequency setting is set to Normal.

SIMCONNECT_DYNAMIC_FREQUENCY DENSE

The current dynamic scenery frequency setting is set to Dense.

SIMCONNECT_DYNAMIC_FREQUENCY VERY_DENSE

The current dynamic scenery frequency setting is set to Very Dense.

SIMCONNECT_DYNAMIC_FREQUENCY EXTREMELY_DENSE

The current dynamic scenery frequency setting is set to Extremely Dense.

Working Samples

Primary samples None.

See Also

- [SimConnect_RequestTrafficSettings](#)
 - [SimConnect_SetTrafficSettings](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_DATA_FACILITY_WAYPOINT

The **SIMCONNECT_DATA_FACILITY_WAYPOINT** structure used to return information on a single waypoint in the facilities cache.

Syntax

```
struct SIMCONNECT_DATA_FACILITY_WAYPOINT : public  
SIMCONNECT_DATA_FACILITY_AIRPORT {  
    float fMagVar;  
};
```

Members

fMagVar

The magnetic variation of the waypoint in degrees.

Remarks

This structure is returned as one element in the [SIMCONNECT_RECV_WAYPOINT_LIST](#) structure. It inherits all the members of the [SIMCONNECT_DATA_FACILITY_AIRPORT](#) structure.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SIMCONNECT_DATA_FACILITY_NDB](#)
 - [SIMCONNECT_DATA_FACILITY_VOR](#)
 - [SIMCONNECT_DATA_FACILITY_TACAN](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_DATA_JOYSTICK_DEVICE_INFO

The **SIMCONNECT_DATA_JOYSTICK_DEVICE_INFO** structure is used to return information about connected joystick devices. The [SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO](#) structure contains an array of these, one for each connected joystick device.

Syntax

```
struct SIMCONNECT_DATA_JOYSTICK_DEVICE_INFO {  
    char szName[128];  
    DWORD dwNumber;  
};
```

Members

szName[128];

The name of the connected device.

dwNumber;

The joystick number of the connected device. This number can be used by the

[SimConnect_MapInputEventToClientEvent](#) function.

Remarks

An array of this structure is part of the [**SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO**](#) structure. Each element in the array provides information on a single connected joystick device.

See the remarks for [SimConnect_RequestJoystickDeviceInfo](#).

See Also

- [SimConnect_RequestJoystickDeviceInfo](#)
 - [SimConnect_MapInputEventToClientEvent](#)
 - [**SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO**](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_DATA_MOBILE_SCENERY_INFO

The **SIMCONNECT_DATA_MOBILE_SCENERY_INFO** structure is used to return identification information about mobile scenery objects in a specified radius from the user vehicle. The [**SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS**](#) structure contains an array of these, one for each mobile scenery object in the specified radius from the user vehicle.

Syntax

```
struct SIMCONNECT_DATA_MOBILE_SCENERY_INFO {  
    char szMobileSceneryName[MAX_PATH];  
    DWORD dwObjectID;  
};
```

Members

szMobileSceneryName[MAX_PATH];

The name of the mobile scenery object as defined in the scenario.

dwObjectID;

The object ID of the mobile scenery object. This ID can be used in the

[SimConnect_RequestMobileSceneryDataByID](#) function to request data about the mobile scenery object with this ID.

Remarks

An array of this structure is part of the [**SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS**](#) structure. Each element in the array provides information on a single mobile scenery object.

See the remarks for [SimConnect_RequestMobileSceneryInRadius](#).

See Also

- [SimConnect_RequestMobileSceneryInRadius](#)
 - [SimConnect_RequestMobileSceneryDataByID](#)
 - [**SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS**](#)
 - [**SIMCONNECT_RECV_MOBILE_SCENERY_DATA**](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_PERIOD

The **SIMCONNECT_PERIOD** enumeration type is used with the [SimConnect_RequestDataOnSimObject](#) call to specify how often data is to be sent to the client.

Syntax

```
enum SIMCONNECT_PERIOD{
    SIMCONNECT_PERIOD_NEVER,
    SIMCONNECT_PERIOD_ONCE,
    SIMCONNECT_PERIOD_VISUAL_FRAME,
    SIMCONNECT_PERIOD_SIM_FRAME,
    SIMCONNECT_PERIOD_SECOND,
};
```

Members

SIMCONNECT_PERIOD_NEVER

Specifies that the data is not to be sent.

SIMCONNECT_PERIOD_ONCE

Specifies that the data should be sent once only. Note that this is not an efficient way of receiving data frequently, use one of the other periods if there is a regular frequency to the data request.

SIMCONNECT_PERIOD_VISUAL_FRAME

Specifies that the data should be sent every visual (rendered) frame.

SIMCONNECT_PERIOD_SIM_FRAME

Specifies that the data should be sent every simulated frame, whether that frame is rendered or not.

SIMCONNECT_PERIOD_SECOND

Specifies that the data should be sent once every second.

Working Samples

Primary samples [Tagged Data](#)

Reference samples [Weather Station](#)

Remarks

Although the period definitions are specific, data is always transmitted at the end of a frame, so even if you have specified that data should be sent every second, the data will actually be transmitted at the end of the frame that comes on or after one second has elapsed.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestDataOnSimObject](#)
-

SIMCONNECT_EXCEPTION

The **SIMCONNECT_EXCEPTION** enumeration type is used with the [SIMCONNECT_RECV_EXCEPTION](#) structure to return information on an error that has occurred.

Syntax

```
enum SIMCONNECT_EXCEPTION{
    SIMCONNECT_EXCEPTION_NONE = 0,
    SIMCONNECT_EXCEPTION_ERROR = 1,
```

```
SIMCONNECT_EXCEPTION_SIZE_MISMATCH = 2,
SIMCONNECT_EXCEPTION_UNRECOGNIZED_ID = 3,
SIMCONNECT_EXCEPTION_UNOPENED = 4,
SIMCONNECT_EXCEPTION_VERSION_MISMATCH = 5,
SIMCONNECT_EXCEPTION_TOO_MANY_GROUPS = 6,
SIMCONNECT_EXCEPTION_NAME_UNRECOGNIZED = 7,
SIMCONNECT_EXCEPTION_TOO_MANY_EVENT_NAMES = 8,
SIMCONNECT_EXCEPTION_EVENT_ID_DUPLICATE = 9,
SIMCONNECT_EXCEPTION_TOO_MANY_MAPS = 10,
SIMCONNECT_EXCEPTION_TOO_MANY_OBJECTS = 11,
SIMCONNECT_EXCEPTION_TOO_MANY_REQUESTS = 12,
SIMCONNECT_EXCEPTION_WEATHER_INVALID_PORT = 13,
SIMCONNECT_EXCEPTION_WEATHER_INVALID_METAR = 14,
SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_GET_OBSERVATION = 15,
SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_CREATE_STATION = 16,
SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_REMOVE_STATION = 17,
SIMCONNECT_EXCEPTION_INVALID_DATA_TYPE = 18,
SIMCONNECT_EXCEPTION_INVALID_DATA_SIZE = 19,
SIMCONNECT_EXCEPTION_DATA_ERROR = 20,
SIMCONNECT_EXCEPTION_INVALID_ARRAY = 21,
SIMCONNECT_EXCEPTION_CREATE_OBJECT_FAILED = 22,
SIMCONNECT_EXCEPTION_LOAD_FLIGHTPLAN_FAILED = 23,
SIMCONNECT_EXCEPTION_OPERATION_INVALID_FOR_OJBECT_TYPE = 24,
SIMCONNECT_EXCEPTION_ILLEGAL_OPERATION = 25,
SIMCONNECT_EXCEPTION_ALREADY_SUBSCRIBED = 26,
SIMCONNECT_EXCEPTION_INVALID_ENUM = 27,
SIMCONNECT_EXCEPTION_DEFINITION_ERROR = 28,
SIMCONNECT_EXCEPTION_DUPLICATE_ID = 29,
SIMCONNECT_EXCEPTION_DATUM_ID = 30,
SIMCONNECT_EXCEPTION_OUT_OF_BOUNDS = 31,
SIMCONNECT_EXCEPTION_ALREADY_CREATED = 32,
SIMCONNECT_EXCEPTION_OBJECT_OUTSIDE_REALITY_BUBBLE = 33,
SIMCONNECT_EXCEPTION_OBJECT_CONTAINER = 34,
SIMCONNECT_EXCEPTION_OBJECT_AI = 35,
SIMCONNECT_EXCEPTION_OBJECT_ATC = 36,
SIMCONNECT_EXCEPTION_OBJECT_SCHEDULE = 37,
SIMCONNECT_EXCEPTION_BLOCK_TIMEOUT = 38,
};
```

Members

General errors

SIMCONNECT_EXCEPTION_NONE

Specifies that there has not been an error. This value is not currently used.

SIMCONNECT_EXCEPTION_ERROR

An unspecific error has occurred. This can be from incorrect flag settings, null or incorrect parameters, the need to have at least one up or down event with an input event, failed calls from the SimConnect server to the operating system, among other reasons.

SIMCONNECT_EXCEPTION_SIZE_MISMATCH

Specifies the size of the data provided does not match the size required. This typically occurs when the wrong string length, fixed or variable, is involved.

SIMCONNECT_EXCEPTION_UNRECOGNIZED_ID

Specifies that the client event, request ID, data definition ID, or object ID was not recognized.

SIMCONNECT_EXCEPTION_UNOPENED

Specifies that communication with the SimConnect server has not been opened. This error is not currently used.

SIMCONNECT_EXCEPTION_VERSION_MISMATCH

Specifies a versioning error has occurred. Typically this will occur when a client built on a newer version of the SimConnect client dll attempts to work with an older version of the SimConnect server.

SIMCONNECT_EXCEPTION_TOO_MANY_GROUPS

Specifies that the maximum number of groups allowed has been reached. The maximum is 20.

SIMCONNECT_EXCEPTION_NAME_UNRECOGNIZED

Specifies that the simulation event name (such as "brakes") is not recognized.

SIMCONNECT_EXCEPTION_TOO_MANY_EVENT_NAMES

Specifies that the maximum number of event names allowed has been reached. The maximum is 1000.

SIMCONNECT_EXCEPTION_EVENT_ID_DUPLICATE

Specifies that the event ID has been used already. This can occur with calls to

[SimConnect_MapClientEventToSimEvent](#), or [SimConnect_SubscribeToSystemEvent](#).

SIMCONNECT_EXCEPTION_TOO_MANY_MAPS

Specifies that the maximum number of mappings allowed has been reached. The maximum is 20.

SIMCONNECT_EXCEPTION_TOO_MANY_OBJECTS

Specifies that the maximum number of objects allowed has been reached. The maximum is 1000.

SIMCONNECT_EXCEPTION_TOO_MANY_REQUESTS

Specifies that the maximum number of requests allowed has been reached. The maximum is 1000.

SIMCONNECT_EXCEPTION_INVALID_DATA_TYPE

Specifies that the data type requested does not apply to the type of data requested. Typically this occurs with a fixed length string of the wrong length.

SIMCONNECT_EXCEPTION_INVALID_DATA_SIZE

Specifies that the size of the data provided is not what is expected. This can occur when the size of a structure provided does not match the size given, or a null string entry is made for a menu or sub-menu entry text, or data with a size of zero is added to a data definition. It can also occur with an invalid request to [SimConnect_CreateClientData](#).

SIMCONNECT_EXCEPTION_DATA_ERROR

Specifies a generic data error. This error is used by the [SimConnect_WeatherCreateThermal](#) function to report incorrect parameters, such as negative radii or values greater than the maximum allowed. It is also used by the [SimConnect_FlightSave](#) and [SimConnect_FlightLoad](#) functions to report incorrect file types. It is also used by other functions to report that flags or reserved parameters have not been set to zero.

SIMCONNECT_EXCEPTION_INVALID_ARRAY

Specifies an invalid array has been sent to the [SimConnect_SetDataOnSimObject](#) function.

SIMCONNECT_EXCEPTION_ALREADY_SUBSCRIBED

Specifies that the client has already subscribed to that event.

SIMCONNECT_EXCEPTION_INVALID_ENUM

Specifies that the member of the enumeration provided was not valid. Currently this is only used if an unknown type is provided to [SimConnect_RequestDataOnSimObjectType](#).

SIMCONNECT_EXCEPTION_DEFINITION_ERROR

Specifies that there is a problem with a data definition. Currently this is only used if a variable length definition is sent with [SimConnect_RequestDataOnSimObject](#).

SIMCONNECT_EXCEPTION_DUPLICATE_ID

Specifies that the ID has already been used. This can occur with menu IDs, or with the IDs provided to [SimConnect_AddToDataDefinition](#), [SimConnect_AddClientEventToNotificationGroup](#) or [SimConnect_MapClientDataNameToID](#).

SIMCONNECT_EXCEPTION_DATUM_ID

Specifies that the datum ID is not recognized. This currently occurs with a call to the [SimConnect_SetDataOnSimObject](#) function.

SIMCONNECT_EXCEPTION_OUT_OF_BOUNDS

Specifies that the radius given in the [SimConnect_RequestDataOnSimObjectType](#) was outside the acceptable range, or with an invalid request to [SimConnect_CreateClientData](#).

SIMCONNECT_EXCEPTION_ALREADY_CREATED

Specifies that a client data area with the name requested by a call to [SimConnect_MapClientDataNameToID](#) has already been created by another add-on. Try again with a different name.

Weather system errors**SIMCONNECT_EXCEPTION_WEATHER_INVALID_PORT**

Specifies an invalid port number was requested.

SIMCONNECT_EXCEPTION_WEATHER_INVALID_METAR

Specifies that the metar data supplied did not match the required format. See the section [Metar Data Format](#) for details on the format required.

SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_GET_OBSERVATION

Specifies that the weather observation requested was not available. Refer to the remarks section for [SimConnect_WeatherRequestObservationAtStation](#) for some notes on this exception.

SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_CREATE_STATION

Specifies that the weather station could not be created.

SIMCONNECT_EXCEPTION_WEATHER_UNABLE_TO_REMOVE_STATION

Specifies that the weather station could not be removed.

AI errors**SIMCONNECT_EXCEPTION_CREATE_OBJECT_FAILED**

Specifies that the attempt to create an AI object failed.

SIMCONNECT_EXCEPTION_LOAD_FLIGHTPLAN_FAILED

Specifies that the specified flight plan could not be found, or did not load correctly.

SIMCONNECT_EXCEPTION_OPERATION_INVALID_FOR_OBJECT_TYPE

Specifies that the operation requested does not apply to the object type, for example trying to set a flight plan on an object that is not an aircraft will result in this error.

SIMCONNECT_EXCEPTION_ILLEGAL_OPERATION

Specifies that the AI operation requested cannot be completed, such as requesting that an object be removed when the client did not create that object.

SIMCONNECT_EXCEPTION_OBJECT_OUTSIDE_REALITY_BUBBLE

Specifies that an attempt to create an ATC controlled AI object failed because the location of the object is outside the [reality bubble](#).

SIMCONNECT_EXCEPTION_OBJECT_CONTAINER

Specifies that an attempt to create an AI object failed because of an error with the container system for the object.

SIMCONNECT_EXCEPTION_OBJECT_AI

Specifies that an attempt to create an AI object failed because of an error with the AI system for the object.

SIMCONNECT_EXCEPTION_OBJECT_ATC

Specifies that an attempt to create an AI object failed because of an error with the ATC system for the object.

SIMCONNECT_EXCEPTION_OBJECT_SCHEDULE

Specifies that an attempt to create an AI object failed because of a scheduling problem.

SIMCONNECT_EXCEPTION_BLOCK_TIMEOUT

Specifies that a synchronous blocking callback didn't release the block within the timeout period.

Working Sample

Primary sample [Tracking Errors](#).

Remarks

In the context of SimConnect, exceptions are error codes, and should not be confused with the C# or system concepts of exceptions. Refer to the remarks for [SimConnect_GetLastSentPacketID](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
-

SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG

The **SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG** enumeration type is used to specify which External Sim Callbacks the client wishes to receive.

Syntax

```
enum SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG {
    SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_CREATE = 0x00000001,
    SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_DESTROY = 0x00000002,
    SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_SIMULATE = 0x00000004,
    SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_LOCATION_CHANGED =
0x00000008,
    SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_EVENT = 0x00000010,
```

Members

SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_CREATE

Set this bit to request External Sim Create Callbacks.

SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_DESTROY

Set this bit to request External Sim Destroy Callbacks.

SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_SIMULATE

Set this bit to request External Sim Simulate Callbacks.

SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_LOCATION_CHANGED

Set this bit to request External Sim Location Changed Callbacks.

SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_EVENT

Set this bit to request External Sim Event Callbacks.

Working Sample

Primary sample [External Sim](#).

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_GOAL_RESOLUTION

The **SIMCONNECT_GOAL_RESOLUTION** enumeration type is used by [SimConnect_ResolveGoal](#) to specify the state the specified goal should be resolved to.

Syntax

```
enum SIMCONNECT_GOAL_RESOLUTION {
    SIMCONNECT_GOAL_RESOLUTION_COMPLETED,
    SIMCONNECT_GOAL_RESOLUTION_FAILED,
};
```

Members

SIMCONNECT_GOAL_RESOLUTION_COMPLETED

The goal will be resolved to Completed.

SIMCONNECT_GOAL_RESOLUTION_FAILED

The goal will be resolved to Failed.

Working Samples

Primary samples [Managed Mission Objects](#)

See Also

- [SimConnect_ResolveGoal](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_GOAL_STATE

The **SIMCONNECT_GOAL_STATE** enumeration type is used inside of the [SIMCONNECT_RECV_GOAL](#) struct to specify the state of the the goal represented by that struct. settings.

Syntax

```
enum SIMCONNECT_GOAL_STATE {
    SIMCONNECT_GOAL_STATE_GOAL_PENDING,
    SIMCONNECT_GOAL_STATE_GOAL_COMPLETED,
    SIMCONNECT_GOAL_STATE_GOAL_FAILED,
};
```

Members

SIMCONNECT_GOAL_STATE_GOAL_PENDING

The state of this goal is Pending.

SIMCONNECT_GOAL_STATE_GOAL_COMPLETED

The state of this goal is Completed.

SIMCONNECT_GOAL_STATE_GOAL_FAILED

The state of this goal is Failed.

Working Samples

Primary samples [Managed Mission Objects](#)

See Also

- [SIMCONNECT_RECV_GOAL](#)
 - [SimConnect_RequestGoalDataByGUID](#)
 - [SimConnect_RequestGoalDataByIndex](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_GROUND_INFO_LATLON_FORMAT

The **SIMCONNECT_GROUND_INFO_LATLON_FORMAT** enumeration type is used with the [SimConnect_RequestGroundInfo](#) and [SimConnect_RequestGroundInfoOnSimObject](#) functions to specify the format for latitude and longitude values.

Syntax

```
enum SIMCONNECT_GROUND_INFO_LATLON_FORMAT {  
    SIMCONNECT_GROUND_INFO_LATLON_FORMAT_RADIANS,  
    SIMCONNECT_GROUND_INFO_LATLON_FORMAT_DEGREES,  
    SIMCONNECT_GROUND_INFO_LATLON_FORMAT_METERS,
```

Members

SIMCONNECT_GROUND_INFO_LATLON_FORMAT_RADIANS

Latitude and Longitude values will be in radians.

SIMCONNECT_GROUND_INFO_LATLON_FORMAT_DEGREES

Latitude and Longitude values will be in degrees.

SIMCONNECT_GROUND_INFO_LATLON_FORMAT_METERS

Latitude and Longitude values will be in meters.

See Also

- [SimConnect_RequestGroundInfo](#)
 - [SimConnect_RequestGroundInfoOnSimObject](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_GROUND_INFO_ALT_FORMAT

The **SIMCONNECT_GROUND_INFO_ALT_FORMAT** enumeration type is used with the [SimConnect_RequestGroundInfo](#) and [SimConnect_RequestGroundInfoOnSimObject](#) functions to specify the format for altitude values.

Syntax

```
enum SIMCONNECT_GROUND_INFO_ALT_FORMAT {  
    SIMCONNECT_GROUND_INFO_ALT_FORMAT_METERS,  
    SIMCONNECT_GROUND_INFO_ALT_FORMAT_FEET,
```

Members

SIMCONNECT_GROUND_INFO_ALT_FORMAT_METERS

Altitude values will be in meters.

SIMCONNECT_GROUND_INFO_ALT_FORMAT_FEET

Altitude values will be in feet.

See Also

- [SimConnect_RequestGroundInfo](#)
-

-
- [SimConnect_RequestGroundInfoOnSimObject](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_GROUND_INFO_SOURCE_FLAG

The **SIMCONNECT_GROUND_INFO_SOURCE_FLAG** enumeration type is used with the [SimConnect_RequestGroundInfo](#) and [SimConnect_RequestGroundInfoOnSimObject](#) functions to specify the format for latitude and longitude values.

Syntax

```
enum SIMCONNECT_GROUND_INFO_SOURCE_FLAG {  
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG_TERRAIN = 0x00010000,  
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG_PLATFORMS = 0x00020000,  
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG_BATHYMETRY = 0x00040000,  
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG_IGNORE_WAVES = 0x00080000,  
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG_TERRAIN_AVG =  
    SIMCONNECT_GROUND_INFO_SOURCE_FLAG_IGNORE_WAVES,  
};
```

Members

SIMCONNECT_GROUND_INFO_SOURCE_FLAG_TERRAIN

Return altitude based on terrain data if encountered first.

SIMCONNECT_GROUND_INFO_SOURCE_FLAG_PLATFORMS

Return altitude based on platform data if encountered first.

SIMCONNECT_GROUND_INFO_SOURCE_FLAG_BATHYMETRY

Return altitude based on bathymetry data if available, water surface will be returned if bathymetry data not available.

SIMCONNECT_GROUND_INFO_SOURCE_FLAG_IGNORE_WAVES

SIMCONNECT_GROUND_INFO_SOURCE_FLAG_TERRAIN_AVG

Return average altitude based on terrain data if encountered first. Returns the average water surface altitude when animated water surfaces are enabled.

See Also

- [SimConnect_RequestGroundInfo](#)
 - [SimConnect_RequestGroundInfoOnSimObject](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_FACILITY_LIST_TYPE

The **SIMCONNECT_FACILITY_LIST_TYPE** enumeration type is used to determine which type of facilities data is being requested or returned.

Syntax

```
enum SIMCONNECT_FACILITY_LIST_TYPE{  
    SIMCONNECT_FACILITY_LIST_TYPE_AIRPORT,  
    SIMCONNECT_FACILITY_LIST_TYPE_WAYPOINT,  
    SIMCONNECT_FACILITY_LIST_TYPE_NDB,  
    SIMCONNECT_FACILITY_LIST_TYPE_VOR,  
    SIMCONNECT_FACILITY_LIST_TYPE_TACAN,  
    SIMCONNECT_FACILITY_LIST_TYPE_COUNT  
};
```

Members

SIMCONNECT_FACILITY_LIST_TYPE_AIRPORT

Specifies that the type of information is for an airport, see [SIMCONNECT_DATA_FACILITY_AIRPORT](#).

SIMCONNECT_FACILITY_LIST_TYPE_WAYPOINT

Specifies that the type of information is for a waypoint, see [SIMCONNECT_DATA_FACILITY_WAYPOINT](#).

SIMCONNECT_FACILITY_LIST_TYPE_NDB

Specifies that the type of information is for an NDB, see [SIMCONNECT_DATA_FACILITY_NDB](#).

SIMCONNECT_FACILITY_LIST_TYPE_VOR

Specifies that the type of information is for a VOR, see [SIMCONNECT_DATA_FACILITY_VOR](#).

SIMCONNECT_FACILITY_LIST_TYPE_TACAN

Specifies that the type of information is for a TACAN, see [SIMCONNECT_DATA_FACILITY_TACAN](#).

SIMCONNECT_FACILITY_LIST_TYPE_COUNT

Not valid as a list type, but simply the number of list types.

Working Sample

Primary sample [FacilitiesData](#)

Remarks

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect_RequestFacilitiesList](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_LICENSE_TYPE

The **SIMCONNECT_LICENSE_TYPE** enumeration type is used to determine which type of Prepar3D license is being used.

Syntax

```
enum SIMCONNECT_LICENSE_TYPE{
    SIMCONNECT_LICENSE_ACADEMIC,
    SIMCONNECT_LICENSE_PROFESSIONAL,
    SIMCONNECT_LICENSE_PROFESSIONAL_PLUS,
    SIMCONNECT_LICENSE_UNKNOWN,
};
```

Members

SIMCONNECT_LICENSE_ACADEMIC

Specifies that the type of license being used is Academic. See [SIMCONNECT_RECV_VERSION](#).

SIMCONNECT_LICENSE_PROFESSIONAL

Specifies that the type of license being used is Professional. See [SIMCONNECT_RECV_VERSION](#).

SIMCONNECT_LICENSE_PROFESSIONAL_PLUS

Specifies that the type of license being used is Professional Plus. See [SIMCONNECT_RECV_VERSION](#).

SIMCONNECT_LICENSE_UNKNOWN

Specifies that the type of license being used could not be determined. See

[SIMCONNECT_RECV_VERSION](#)

Remarks

See the remarks for [SimConnect_RequestVersion](#).

See Also

- [SimConnect_RequestVersion](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_MISSION_END

The **SIMCONNECT_MISSION_END** enumeration type is used to specify the three possible outcomes of a scenario.

Syntax

```
enum SIMCONNECT_MISSION_END{
    SIMCONNECT_MISSION_FAILED,
    SIMCONNECT_MISSION_CRASHED,
    SIMCONNECT_MISSION_SUCCEEDED
};
```

Members

SIMCONNECT_MISSION_FAILED

Specifies that the user did not complete the scenario successfully.

SIMCONNECT_MISSION_CRASHED

Specifies that the user aircraft crashed during the scenario.

SIMCONNECT_MISSION_SUCCEEDED

Specifies that the user completed the scenario successfully.

Working Sample

Primary sample [Mission Action](#)

Remarks

Refer to the scenario events that can be requested by the [SimConnect_SubscribeToSystemEvent](#) function. Also see the remarks for [SimConnect_CompleteCustomMissionAction](#).

See Also

- [SimConnect API Reference](#)
-

SIMCONNECT_TEXT_ORIGIN

The **SIMCONNECT_TEXT_ORIGIN** enumeration type is used inside of the [SIMCONNECT_RECV_EVENT_TEXT](#) struct to specify the origin of the text window.

Syntax

```
enum SIMCONNECT_TEXT_ORIGIN{
    SIMCONNECT_TEXT_ORIGIN_APPLICATION,
    SIMCONNECT_TEXT_ORIGIN_SIMCONNECT,
};
```

Members

SIMCONNECT_TEXT_ORIGIN_APPLICATION

Specifies that the text window was created in Prepar3D.

SIMCONNECT_TEXT_ORIGIN_SIMCONNECT

Specifies that the text window was created in a SimConnect client.

Working Sample

Primary sample [Text Menu](#)

See Also

- [SIMCONNECT_RECV_EVENT_TEXT](#)
 - [SimConnect_SubscribeToSystemEvent](#)
-

SIMCONNECT_MISISON_OBJECT_TYPE

The **SIMCONNECT_MISISON_OBJECT_TYPE** enumeration type is used inside of the [SIMCONNECT_RECV_MISISON_OBJECT_COUNT](#) struct to specify the object type the count is associated with. settings.

Syntax

```
enum SIMCONNECT_MISISON_OBJECT_TYPE {
    SIMCONNECT_MISISON_OBJECT_TYPE_GOAL,
    SIMCONNECT_MISISON_OBJECT_TYPE_MISISON_OBJECTIVE,
    SIMCONNECT_MISISON_OBJECT_TYPE_FLIGHT_SEGMENT,
    SIMCONNECT_MISISON_OBJECT_TYPE_LANDING_TRIGGER
};
```

Members

SIMCONNECT_MISISON_OBJECT_TYPE_GOAL

This count is of Goal/Group Goal scenario objects.

SIMCONNECT_MISISON_OBJECT_TYPE_MISISON_OBJECTIVE

This count is of Mission Objective scenario objects.

SIMCONNECT_MISISON_OBJECT_TYPE_FLIGHT_SEGMENT

This count is of Flight Segment scenario objects.

SIMCONNECT_MISISON_OBJECT_TYPE_LANDING_TRIGGER

This count is of Area or Airport Landing Trigger scenario objects.

Working Samples

Primary samples [Managed Mission Objects](#)

See Also

- [SIMCONNECT_RECV_MISISON_OBJECT_COUNT](#)
 - [SimConnect_RequestFlightSegmentCount](#)
 - [SimConnect_RequestGoalCount](#)
 - [SimConnect_RequestMissionObjectiveCount](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_LANDING_TYPE

The **SIMCONNECT_LANDING_TYPE** enumeration type is used inside of the [SIMCONNECT_RECV_LANDING_INFO](#) struct to specify the type of landing that the landing trigger accepts.

Syntax

```
enum SIMCONNECT_LANDING_TYPE {  
    SIMCONNECT_LANDING_TYPE_ANY,  
    SIMCONNECT_LANDING_TYPE_FULL_STOP,  
    SIMCONNECT_LANDING_TYPE_TOUCHDOWN  
};
```

Members

SIMCONNECT_LANDING_TYPE_ANY

This landing trigger accepts either full stop or touchdown landings.

SIMCONNECT_LANDING_TYPE_FULL_STOP

This landing trigger accepts full stop landings.

SIMCONNECT_LANDING_TYPE_TOUCHDOWN

This landing trigger accepts touchdown landings.

See Also

- [SIMCONNECT_RECV_LANDING_INFO](#)
 - [SimConnect_RequestLandingTriggerLandingInfoByIndex](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_MISISON_OBJECTIVE_STATUS

The **SIMCONNECT_MISISON_OBJECTIVE_STATUS** enumeration type is used inside of the [SIMCONNECT_RECV_MISISON_OBJECTIVE](#) struct to specify the state of the mission objective represented by that struct. settings.

Syntax

```
enum SIMCONNECT_MISISON_OBJECTIVE_STATUS {  
    SIMCONNECT_MISISON_OBJECTIVE_STATUS_PENDING,  
    SIMCONNECT_MISISON_OBJECTIVE_STATUS_PASSED,  
    SIMCONNECT_MISISON_OBJECTIVE_STATUS_FAILED,  
};
```

Members

SIMCONNECT_MISISON_OBJECTIVE_STATUS_PENDING

The state of this mission objective is Pending.

SIMCONNECT_MISISON_OBJECTIVE_STATUS_PASSED

The state of this mission objective is Completed.

SIMCONNECT_MISISON_OBJECTIVE_STATUS_FAILED

The state of this mission objective is Failed.

Working Samples

Primary samples [Managed Mission Objects](#)

See Also

- [SIMCONNECT_RECV_MISSION_OBJECTIVE](#)
 - [SimConnect_RequestMissionObjectiveDataByGUID](#)
 - [SimConnect_RequestMissionObjectiveDataByIndex](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_SCENERY_COMPLEXITY

The **SIMCONNECT_SCENERY_COMPLEXITY** enumeration type is used with the [SimConnect_RequestSceneryComplexity](#) to request the current scenery complexity setting.

Syntax

```
enum SIMCONNECT_SCENERY_COMPLEXITY {  
    SIMCONNECT_SCENERY_COMPLEXITY_VERY_SPARSE,  
    SIMCONNECT_SCENERY_COMPLEXITY_SPARSE,  
    SIMCONNECT_SCENERY_COMPLEXITY_NORMAL,  
    SIMCONNECT_SCENERY_COMPLEXITY_DENSE,  
    SIMCONNECT_SCENERY_COMPLEXITY_VERY_DENSE,  
    SIMCONNECT_SCENERY_COMPLEXITY_EXTREMELY_DENSE,  
};
```

Members

SIMCONNECT_SCENERY_COMPLEXITY_VERY_SPARSE

The current scenery complexity setting is set to Very Sparse.

SIMCONNECT_SCENERY_COMPLEXITY_SPARSE

The current scenery complexity setting is set to Sparse.

SIMCONNECT_SCENERY_COMPLEXITY_NORMAL

The current scenery complexity setting is set to Normal.

SIMCONNECT_SCENERY_COMPLEXITY_DENSE

The current scenery complexity setting is set to Dense.

SIMCONNECT_SCENERY_COMPLEXITY_VERY_DENSE

The current scenery complexity setting is set to Very Dense.

SIMCONNECT_SCENERY_COMPLEXITY_EXTREMELY_DENSE

The current scenery complexity setting is set to Extremely Dense.

Working Samples

Primary samples [Scenery Complexity and Shadow Flags](#)

See Also

- [SimConnect_RequestSceneryComplexity](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_SHADOW_FLAGS

The **SIMCONNECT_SHADOW_FLAGS** enumeration type is used with the [SimConnect_RequestShadowFlags](#) to request the current shadow flag settings.

Syntax

```
enum SIMCONNECT_SHADOW_FLAGS {  
    SIMCONNECT_SHADOW_FLAGS_INTERIOR_CAST = 0x00000001,  
    SIMCONNECT_SHADOW_FLAGS_INTERIOR_RECEIVE = 0x00000002,
```

```
SIMCONNECT_SHADOW_FLAGS_EXTERIOR_CAST      = 0x00000004,
SIMCONNECT_SHADOW_FLAGS_EXTERIOR_RECEIVE    = 0x00000008,
SIMCONNECT_SHADOW_FLAGS_SIMOBJECTS_CAST     = 0x00000010,
SIMCONNECT_SHADOW_FLAGS_SIMOBJECTS_RECEIVE  = 0x00000020,
SIMCONNECT_SHADOW_FLAGS_TERRAIN_CAST        = 0x00000040,
SIMCONNECT_SHADOW_FLAGS_TERRAIN_RECEIVE     = 0x00000080,
SIMCONNECT_SHADOW_FLAGS_VEGETATION_CAST     = 0x00000100,
SIMCONNECT_SHADOW_FLAGS_VEGETATION_RECEIVE  = 0x00000200,
SIMCONNECT_SHADOW_FLAGS_BUILDINGS_CAST      = 0x00000400,
SIMCONNECT_SHADOW_FLAGS_BUILDINGS_RECEIVE   = 0x00000800,
SIMCONNECT_SHADOW_FLAGS_CLOUDS_CAST         = 0x00001000,
};
```

Members

SIMCONNECT_SHADOW_FLAGS_INTERIOR_CAST

The user interior model is casting shadows.

SIMCONNECT_SHADOW_FLAGS_INTERIOR_RECEIVE

The user interior model is receiving shadows.

SIMCONNECT_SHADOW_FLAGS_EXTERIOR_CAST

The user exterior model is casting shadows.

SIMCONNECT_SHADOW_FLAGS_EXTERIOR_RECEIVE

The user exterior model is receiving shadows.

SIMCONNECT_SHADOW_FLAGS_SIMOBJECTS_CAST

Other SimObjects are casting shadows.

SIMCONNECT_SHADOW_FLAGS_SIMOBJECTS_RECEIVE

Other SimObjects are receiving shadows.

SIMCONNECT_SHADOW_FLAGS_TERRAIN_CAST

The terrain is casting shadows.

SIMCONNECT_SHADOW_FLAGS_TERRAIN_RECEIVE

The terrain is receiving shadows.

SIMCONNECT_SHADOW_FLAGS_VEGETATION_CAST

Autogen vegetation are casting shadows.

SIMCONNECT_SHADOW_FLAGS_VEGETATION_RECEIVE

Autogen vegetation are receiving shadows.

SIMCONNECT_SHADOW_FLAGS_BUILDINGS_CAST

Autogen buildings are casting shadows.

SIMCONNECT_SHADOW_FLAGS_BUILDINGS_RECEIVE

Autogen buildings are receiving shadows.

SIMCONNECT_SHADOW_FLAGS_CLOUDS_CAST

Clouds are casting shadows.

Working Samples

Primary samples [Scenery Complexity and Shadow Flags](#)

See Also

- [SimConnect_RequestShadowFlags](#)
- [SimConnect API Reference](#)

SIMCONNECT_RECV

The **SIMCONNECT_RECV** structure is used with the [SIMCONNECT_RECV_ID](#) enumeration to indicate which type of structure has been returned.

Syntax

```
struct SIMCONNECT_RECV{
    DWORD dwSize;
```

```
    DWORD dwVersion;
    DWORD dwID;
};
```

Members

dwSize

The total size of the returned structure in bytes (that is, not usually the size of the **SIMCONNECT_RECV** structure, but of the structure that inherits it).

dwVersion

The version number of the SimConnect server.

dwID

The ID of the returned structure. One member of [SIMCONNECT_RECV_ID](#).

Working Samples

Primary samples	Client Event No Callback Tracking Errors
------------------------	--

Reference samples All but a few of the other [samples](#) implement this structure.

Remarks

This structure is used as a base class for all other receive data structures. See [SIMCONNECT_RECV_ID](#) for list of possible values for dwID and corresponding SIMCONNECT_RECV_XXX structure used.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_AIRPORT_LIST

The **SIMCONNECT_RECV_AIRPORT_LIST** structure is used to return a list of [SIMCONNECT_DATA_FACILITY_AIRPORT](#) structures.

Syntax

```
struct SIMCONNECT_RECV_AIRPORT_LIST : public
SIMCONNECT_RECV_FACILITIES_LIST{
    SIMCONNECT_DATA_FACILITY_AIRPORT rgData[1];
};
```

Members

rgData[1]

Array of [SIMCONNECT_DATA_FACILITY_AIRPORT](#) structures.

Remarks

This structure inherits the [SIMCONNECT_RECV_FACILITIES_LIST](#) structure, which identifies the number of elements in the list, and the number of packets needed to transmit all the data.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

-
- [SimConnect_RequestFacilitiesList](#)
 - [SIMCONNECT_RECV_FACILITIES_LIST](#)
 - [SIMCONNECT_RECV_NDB_LIST](#)
 - [SIMCONNECT_RECV_VOR_LIST](#)
 - [SIMCONNECT_RECV_TACAN_LIST](#)
 - [SIMCONNECT_RECV_WAYPOINT_LIST](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_ASSIGNED_OBJECT_ID

The **SIMCONNECT_RECV_ASSIGNED_OBJECT_ID** structure is used to return an object ID that matches a request ID.

Syntax

```
struct SIMCONNECT_RECV_ASSIGNED_OBJECT_ID : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwObjectID;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwObjectID

Double word containing the server defined object ID.

Working Samples

[AI Objects and Waypoints](#)

[AI Traffic](#)

Primary samples

[Managed AI Waypoints](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_ASSIGNED_OBJECT_ID**.

See Also

- [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_RECV_SIMOBJECT_DATA](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_ATTACHPOINT_DATA

The **SIMCONNECT_RECV_ATTACHPOINT_DATA** structure will be received by the client after a successful call to [SimConnect_RequestAttachPointData](#).

Syntax

```
struct SIMCONNECT_RECV_ATTACHPOINT_DATA : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwObjectID;  
    SIMCONNECT_DATA_XYZ xyzLocalOffset;  
    SIMCONNECT_DATA_PBH pbhLocalRotation;
```

```
SIMCONNECT_DATA_LATLONALT llaWorldPosition;
SIMCONNECT_DATA_PBH     pbhWorldRotation;
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwObjectID

Double word containing the server defined object ID.

xyzLocalOffset

A [SIMCONNECT_DATA_XYZ](#) containing the local offset of the attach point in feet.

pbhLocalRotation

A [SIMCONNECT_DATA_PBH](#) containing the local rotation of the attach point in radians.

llaWorldPosition

A [SIMCONNECT_DATA_LATLONALT](#) containing the world position of the attach point in radians (altitude in feet).

pbhWorldRotation

A [SIMCONNECT_DATA_PBH](#) containing the world rotation of the attach point in radians.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ATTACHPOINT_DATA](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect_RequestAttachPointData](#)
-

SIMCONNECT_RECV_CAMERA_6DOF

The [SIMCONNECT_RECV_CAMERA_6DOF](#) structure will be received by the client after a successful call to [SimConnect_RequestCameraRelative6DOF](#) or [SimConnect_RequestCameraRelative6DofByName](#).

Syntax

```
struct SIMCONNECT_RECV_CAMERA_6DOF : public SIMCONNECT_RECV {
    DWORD dwRequestID;
    float fDeltaXMeters;
    float fDeltaYMeters;
    float fDeltaZMeters;
    float fPitchDeg;
    float fBankDeg;
    float fHeadingDeg;
};
```

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_CAMERA_6DOF](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_CAMERA_FOV

The **SIMCONNECT_RECV_CAMERA_FOV** structure will be received by the client after a successful call to [SimConnect_RequestCameraFov](#) or [SimConnect_RequestMainCameraFov](#).

Syntax

```
struct SIMCONNECT_RECV_CAMERA_FOV : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    float fHorizontalFov;  
    float fVerticalFov;  
};
```

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_CAMERA_FOV**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_CAMERA_SENSOR_MODE

The **SIMCONNECT_RECV_CAMERA_SENSOR_MODE** structure will be received by the client after a successful call to [SimConnect_RequestCameraSensorMode](#) or [SimConnect_RequestMainCameraSensorMode](#).

Syntax

```
struct SIMCONNECT_RECV_CAMERA_SENSOR_MODE : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    SIMCONNECT_CAMERA_SENSOR_MODE eSensorMode;  
};
```

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_CAMERA_SENSOR_MODE**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_CAMERA_SENSOR_MODE](#)
-

SIMCONNECT_RECV_CAMERA_WINDOW_POSITION

The **SIMCONNECT_RECV_CAMERA_WINDOW_POSITION** structure will be received by the client after a successful call to [SimConnect_RequestCameraWindowPosition](#).

Syntax

```
struct SIMCONNECT_RECV_CAMERA_WINDOW_POSITION : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwXPosition;  
    DWORD dwYPosition;
```

};

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_CAMERA_WINDOW_POSITION](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_CAMERA_WINDOW_SIZE

The [SIMCONNECT_RECV_CAMERA_WINDOW_SIZE](#) structure will be received by the client after a successful call to [SimConnect_RequestCameraWindowSize](#).

Syntax

```
struct SIMCONNECT_RECV_CAMERA_WINDOW_SIZE : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwWidth;  
    DWORD dwHeight;  
};
```

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_CAMERA_WINDOW_SIZE](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_CLIENT_DATA

The [SIMCONNECT_RECV_CLIENT_DATA](#) structure will be received by the client after a successful call to [SimConnect_RequestClientData](#). It is an identical structure to [SIMCONNECT_RECV_SIMOBJECT_DATA](#).

Syntax

```
struct SIMCONNECT_RECV_CLIENT_DATA : public  
SIMCONNECT_RECV_SIMOBJECT_DATA {  
};
```

Remarks

This structure inherits the [SIMCONNECT_RECV_SIMOBJECT_DATA](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_CLIENT_DATA](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_RECV_SIMOBJECT_DATA](#)
-

-
- [SimConnect_RequestClientData](#)
-

SIMCONNECT_RECV_CLOUD_STATE

The **SIMCONNECT_RECV_CLOUD_STATE** structure is used to return an array of cloud state data.

Syntax

```
struct SIMCONNECT_RECV_CLOUD_STATE : public SIMCONNECT_RECV {  
    DWORD dwRequestId;  
    DWORD dwArraySize;  
    BYTE rgbData[1];  
};
```

Members

dwRequestId

Double word containing the client defined request ID.

dwArraySize

Double word starting the cloud data array. The array will be 64 x 64 bytes in size, and each byte will contain a value indicating the cloud density for each cell. A value of zero would mean no clouds, to a maximum of 255. The size of each cell is determined by a call to

[SimConnect_WeatherRequestCloudState](#).

rgbData[1]

Byte array containing the cloud data.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_CLOUD_STATE**.

See Also

- [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_RECV_WEATHER_OBSERVATION](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_CUSTOM_ACTION

The **SIMCONNECT_RECV_CUSTOM_ACTION** structure is used specifically with the mission system, providing details on the custom action that has been triggered.

Syntax

```
struct SIMCONNECT_RECV_CUSTOM_ACTION : public SIMCONNECT_RECV_EVENT {  
    GUID guidInstanceId;  
    DWORD dwWaitForCompletion;  
    char szPayLoad[1];  
};
```

Members

guidInstanceId

GUID of the action that executed.

dwWaitForCompletion

The value of the Wait-for-completion flag on the action.

szPayLoad

A variable length string that is defined in the scenario object XML file. It is specified by the scenario designer and can contain anything that the client might find useful.

Working Sample

Primary sample [Mission Action](#)

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure, which inherits the [SIMCONNECT_RECV](#) structure, and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to **SIMCONNECT_RECV_ID_CUSTOM_ACTION**.

See the remarks for [SimConnect CompleteCustomMissionAction](#).

See Also

- [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT

The **SIMCONNECT_RECV_EVENT** structure is used to return an event ID to the client.

Syntax

```
struct SIMCONNECT_RECV_EVENT : public SIMCONNECT_RECV {  
    DWORD uGroupId;  
    DWORD uEventId;  
    DWORD dwData;  
};
```

Members

uGroupId

The ID of the client defined group, or the special case value: **UNKNOWN_GROUP** (which equals **DWORD_MAX**).

uEventId

The ID of the client defined event that has been requested (such as **EVENT_1** or **EVENT_BRAKES** from the examples in this document).

dwData

This value is usually zero, but some events require further qualification. For example, joystick movement events require a movement value in addition to the notification that the joystick has been moved (see [SimConnect_MapInputEventToClientEvent](#) for more information).

Working Samples

[Client Event](#)
[Cockpit Camera](#)
[Input Event](#)
[Joystick Input](#)
[Menu Items](#)
[No Callback](#)
[Send Event A](#)
[Send Event B](#)
[Send Event C](#)
[Throttle Control](#)
[Tracking Errors](#)

Primary samples

Reference samples All but a few of the other [samples](#) implement this structure.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_EVENT](#). This structure is inherited by several other structures:

- [SIMCONNECT_RECV_CUSTOM_ACTION](#)
- [SIMCONNECT_RECV_EVENT_FILENAME](#)
- [SIMCONNECT_RECV_EVENT_FRAME](#)
- [SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE](#)
- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED](#)
- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED](#)
- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED](#)

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_EVENT_64

The **SIMCONNECT_RECV_EVENT_64** structure is used to return an event ID along with 64-bit user context data to the client.

Syntax

```
struct SIMCONNECT_RECV_EVENT_64 : public SIMCONNECT_RECV_EVENT {  
    QWORD qwData;  
};
```

Members

qwData

User context data provided by 64-bit version functions. (see [SimConnect_TransmitClientEvent64](#), [SimConnect_MenuAddItem64](#), and [SimConnect_MenuAddSubItem64](#) for more information).

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_EVENT_64](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_EVENT](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_EVENT_FILENAME

The **SIMCONNECT_RECV_EVENT_FILENAME** structure is used to return a filename and an event ID to the client.

Syntax

```
struct SIMCONNECT_RECV_EVENT_FILENAME : public SIMCONNECT_RECV_EVENT {
    char szFileName[MAX_PATH];
    DWORD dwFlags;
};
```

Members

szFileName[MAX_PATH]

The returned filename.

dwFlags

Reserved, should be 0.

Working Sample

Primary sample [System Event](#).

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure, and is used to attach a filename to the returned event. When the [SIMCONNECT_RECV](#) structure **dwID** parameter is set to [SIMCONNECT_RECV_ID_EVENT_FILENAME](#), this structure is returned.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EVENT](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_EVENT_FILENAME_W

The [SIMCONNECT_RECV_EVENT_FILENAME_W](#) structure is used to return a filename and an event ID to the client.

Syntax

```
struct SIMCONNECT_RECV_EVENT_FILENAME_W : public SIMCONNECT_RECV_EVENT
{
    wchar_t szFileName[MAX_PATH];
    DWORD dwFlags;
};
```

Members

szFileName[MAX_PATH]

The returned filename.

dwFlags

Reserved, should be 0.

Working Sample

Primary sample [System Event](#).

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure, and is used to attach a filename

to the returned event. When the [SIMCONNECT_RECV](#) structure **dwID** parameter is set to **SIMCONNECT_RECV_ID_EVENT_FILENAME_W**, this structure is returned.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EVENT](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_EVENT_FRAME

The **SIMCONNECT_RECV_EVENT_FRAME** structure is used with the [SimConnect_SubscribeToSystemEvent](#) call to return the frame rate and simulation speed to the client.

Syntax

```
struct SIMCONNECT_RECV_EVENT_FRAME : public SIMCONNECT_RECV_EVENT {  
    float fFrameRate;  
    float fSimSpeed;  
};
```

Members

fFrameRate

The visual frame rate in frames per second.

fSimSpeed

The simulation rate. For example if the simulation is running at four times normal speed -- 4X -- then 4.0 will be returned.

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure, which inherits the [SIMCONNECT_RECV](#) structure, and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_EVENT_FRAME**. Set the requested system event to "Frame" or "PauseFrame" with the [SimConnect_SubscribeToSystemEvent](#) function to receive this data.

See Also

- [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED

The **SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED** structure is sent to a client when they have successfully joined a multiplayer race.

Syntax

```
struct SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED : public  
SIMCONNECT_RECV_EVENT {};
```

Members

This structure takes no parameters in addition to those inherited from the

[**SIMCONNECT_RECV_EVENT**](#) structure.

Remarks

This event is not transmitted to the host of the session, only to the client that has joined in.

To receive these events, refer to the [SimConnect_SubscribeToSystemEvent](#) function.

See Also

- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED](#)
 - [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED](#)
 - [SIMCONNECT_RECV_EVENT_RACE_END](#)
 - [SIMCONNECT_RECV_EVENT_RACE_LAP](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED

The **SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED** structure is sent to the host when the session is visible to other users in the lobby.

Syntax

```
struct SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED : public  
SIMCONNECT_RECV_EVENT {};
```

Members

This structure takes no parameters in addition to those inherited from the [SIMCONNECT_RECV_EVENT](#) structure.

Remarks

This event is sent only to the host of the session.

See Also

- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED](#)
 - [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED](#)
 - [SIMCONNECT_RECV_EVENT_RACE_END](#)
 - [SIMCONNECT_RECV_EVENT_RACE_LAP](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED

The **SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED** structure is sent to a client when they have requested to leave a race, or to all players when the session is terminated by the host.

Syntax

```
struct SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED : public  
SIMCONNECT_RECV_EVENT {};
```

Members

This structure takes no parameters in addition to those inherited from the [SIMCONNECT_RECV_EVENT](#) structure.

Remarks

This is the only event that is broadcast to all the players in a multiplayer race, in the situation where the host terminates, or simply leaves, the race. If a client ends their own participation in the race, they will be the only one to receive the event.

See Also

- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED](#)
 - [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED](#)
 - [SIMCONNECT_RECV_EVENT_RACE_END](#)
 - [SIMCONNECT_RECV_EVENT_RACE_LAP](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE

The [SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE](#) structure is used to return the type and ID of an AI object that has been added or removed from the simulation, by any client.

Syntax

```
struct SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE : public  
SIMCONNECT_RECV_EVENT {  
    SIMCONNECT_SIMOBJECT_TYPE eObjType;  
};
```

Members

eObjType

Specifies the type of object that was added or removed. One member of the [SIMCONNECT_SIMOBJECT_TYPE](#) enumeration.

Working Sample

Primary sample [AI Traffic](#)

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure, which inherits the [SIMCONNECT_RECV](#) structure, and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_EVENT_ADDREMOVE](#). A client can determine whether the object was added or removed from its own event ID that was provided as a parameter to the [SimConnect_SubscribeToSystemEvent](#) function.

The ID of the object added or removed is returned in the dwData parameter (a member of the [SIMCONNECT_RECV_EVENT](#) structure).

See Also

- [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_RACE_END

The **SIMCONNECT_RECV_EVENT_RACE_END** structure is used in multiplayer racing to hold the results for one player at the end of a race.

Syntax

```
struct SIMCONNECT_RECV_EVENT_RACE_END : public SIMCONNECT_RECV_EVENT {  
    DWORD dwRacerNumber;  
    SIMCONNECT_DATA_RACE_RESULT RacerData;  
};
```

Members

dwRacerNumber

The index of the racer the results are for. Players are indexed from 0.

RacerData

A [SIMCONNECT_DATA_RACE_RESULT](#) structure.

Remarks

In a multiplayer race players can come and go, so index numbers are not a reliable means of identifying the players. The **szPlayerName** and **szPlayerRole** parameters of the [SIMCONNECT_DATA_RACE_RESULT](#) structure should be used to identify each player.

See Also

- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED](#)
 - [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED](#)
 - [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED](#)
 - [SIMCONNECT_RECV_EVENT_RACE_LAP](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_RACE_LAP

The **SIMCONNECT_RECV_EVENT_RACE_LAP** structure is used in multiplayer racing to hold the results for one player at the end of a lap.

Syntax

```
struct SIMCONNECT_RECV_EVENT_RACE_LAP : public SIMCONNECT_RECV_EVENT {  
    DWORD dwLapIndex;  
    SIMCONNECT_DATA_RACE_RESULT RacerData;  
};
```

Members

dwLapIndex

The index of the lap the results are for. Laps are indexed from 0.

RacerData

A [SIMCONNECT_DATA_RACE_RESULT](#) structure.

Remarks

None.

See Also

- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED](#)
-

-
- [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED](#)
 - [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED](#)
 - [SIMCONNECT_RECV_EVENT_RACE_END](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EXCEPTION

The **SIMCONNECT_RECV_EXCEPTION** structure is used with the [SIMCONNECT_EXCEPTION](#) enumeration type to return information on an error that has occurred.

Syntax

```
struct SIMCONNECT_RECV_EXCEPTION : public SIMCONNECT_RECV {  
    DWORD dwException;  
    DWORD dwSendID;  
    DWORD dwIndex;  
};
```

Members

dwException

One member of the [SIMCONNECT_EXCEPTION](#) enumeration type, indicating which error has occurred.

dwSendID

The ID of the packet that contained the error, see Remarks below.

dwIndex

The index number (starting at 1) of the first parameter that caused an error. Special case:

UNKNOWN_INDEX = 0.

Working Samples

[Cockpit Camera](#)

Primary samples [Tracking Errors](#)

[Variable Strings](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_EXCEPTION](#).

In order to match the **dwSendID** parameter returned here, with the ID of a request, use the [SimConnect_GetLastSentPacketID](#) call after each request is made.

Note that the HRESULT errors returned after each API call do not involve any communication with the SimConnect server, but are simply client-side errors that are returned immediately. Test for exceptions to check for server-side errors.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EVENT](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_RECV_SIMOBJECT_DATA](#)
-

SIMCONNECT_RECV_EXTERNAL_SIM_BASE

The **SIMCONNECT_RECV_EXTERNAL_SIM_BASE** structure is the base structure for all External Sim related structures. Provides the GUID and SimObjectID for this callback.

Syntax

```
struct SIMCONNECT_RECV_EXTERNAL_SIM_BASE : public SIMCONNECT_RECV{
    GUID guidExternalSimID;
    DWORD dwObjectID;
};
```

Members

guidExternalSimID

GUID identifying the external sim that should handle this callback.

dwObjectID

Double word containing the ID of the SimObject this callback is related to.

Remarks

This structure is inherited by [SIMCONNECT_RECV_EXTERNAL_SIM_CREATE](#), [SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY](#), [SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE](#), [SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED](#), and [SIMCONNECT_RECV_EXTERNAL_SIM_EVENT](#).

This structure inherits the [SIMCONNECT_RECV](#) structure, so use the [SIMCONNECT_RECV_ID](#) enumeration to determine which external sim callback structure has been received.

See the [External Sim Overview](#) for more information.

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_EXTERNAL_SIM_CREATE

The **SIMCONNECT_RECV_EXTERNAL_SIM_CREATE** structure is used with the External Sim Create callback when a new SimObject is being created that uses an external sim.

Syntax

```
struct SIMCONNECT_RECV_EXTERNAL_SIM_CREATE : public
SIMCONNECT_RECV_EXTERNAL_SIM_BASE{
    DWORD dwExternalSimVarCount;
    DWORD dwExternalSimVarBase
    char szExternalSimData[1];
};
```

Members

dwExternalSimVarCount

Double word containing the number of External Sim Vars defined in the vehicles sim.cfg file.

dwExternalSimVarBase

Double word containing the base External Sim Var index for this external sim (only used by Secondary External Sims).

szExternalSimData

Null terminated string containing the data provided in the vehicles sim.cfg file.

Remarks

This structure inherits the [SIMCONNECT_RECV_EXTERNAL_SIM_BASE](#) structure and adds the additional values used by the External Sim Create Callback.

See the [External Sim Overview](#) for more information.

Working Sample

Primary sample [External Sim](#)

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY

The **SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY** structure is used with the External Sim Destroy callback when a SimObject that uses an external sim is being destroyed.

Syntax

```
struct SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY : public  
SIMCONNECT_RECV_EXTERNAL_SIM_BASE{  
    // no callback specific data  
};
```

Members

No callback specific data defined at this time.

Remarks

This structure inherits the [SIMCONNECT_RECV_EXTERNAL_SIM_BASE](#) structure and adds the additional values used by the External Sim Destroy Callback.

See the [External Sim Overview](#) for more information.

Working Sample

Primary sample [External Sim](#)

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE

The **SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE** structure is used with the External Sim Simulate callback when a SimObject that uses an external sim is being simulated.

Syntax

```
struct SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE : public
```

```
SIMCONNECT_RECV_EXTERNAL_SIM_BASE{
    double fDeltaTime;
    BOOL bShouldSimulate;
    DWORD dwDefineID;
    DWORD dwDefineCount;
    DWORD dwData;
};
```

Members

fDeltaTime

Double floating point value specifying the time-delta for this sim frame

bShouldSimulate

Boolean value that denotes whether the external sim should actually simulate this frame or not (don't simulate when this is false).

dwDefineID

Double word containing the data structure definition ID for the data starting at **dwData**.

dwDefineCount

Double word containing the number of Datums (not bytes) being returned

dwData

Start of the returned data structure, take the address of dwData and cast it to an appropriate pointer.

Remarks

This structure inherits the [SIMCONNECT_RECV_EXTERNAL_SIM_BASE](#) structure and adds the additional values used by the External Sim Simulate Callback.

See the [External Sim Overview](#) for more information.

Working Sample

[Primary sample External Sim](#)

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED

The **SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED** structure is used with the External Sim Location Changed callback when a SimObject that uses an external sim has been moved/relocated.

Syntax

```
struct SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED : public
SIMCONNECT_RECV_EXTERNAL_SIM_BASE{
    double fLatRadians;
    double fLonRadians;
    double fAltMeters;
    BOOL bPlaceOnGround;
    BOOL bZeroSpeed;
};
```

Members

fLatRadians

Double floating point value containing new locations Latitude value in radians.

fLonRadians

Double floating point value containing new locations Longitude value in radians.

fAltMeters

Double floating point value containing new locations Altitude value in meters.

bPlaceOnGround

Boolean value, true if vehicle should be force placed onto the ground.

bZeroSpeed

Boolean value, true if all vehicle accelerations/velocities should be reset to zero.

Remarks

This structure inherits the [SIMCONNECT_RECV_EXTERNAL_SIM_BASE](#) structure and adds the additional values used by the External Sim Location Changed Callback.

See the [External Sim Overview](#) for more information.

Working Sample

Primary sample [External Sim](#)

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_EXTERNAL_SIM_EVENT

The **SIMCONNECT_RECV_EXTERNAL_SIM_EVENT** structure is used with the External Sim Event callback when a SimObject that uses an external sim receives an event. The events the external sim client wishes to receive are mapped by calls to the [SimConnect_MapClientEventToSimEvent](#) function.

Syntax

```
struct SIMCONNECT_RECV_EXTERNAL_SIM_EVENT : public  
SIMCONNECT_RECV_EXTERNAL_SIM_BASE{  
    DWORD uEventID;  
    DWORD dwData  
};
```

Members**uEventID**

Double word containing the client defined event ID for this event.

dwExternalSimVarBase

Double word containing any data associated with this event.

Remarks

This structure inherits the [SIMCONNECT_RECV_EXTERNAL_SIM_BASE](#) structure and adds the additional values used by the External Sim Event Callback.

See the [External Sim Overview](#) for more information.

Working Sample

Primary sample [External Sim](#)

See Also

- [SimConnect_RegisterExternalSim](#)
 - [SimConnect_RegisterExternalSecondarySim](#)
 - [SimConnect_MapClientEventToSimEvent](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_EVENT_TEXT

The **SIMCONNECT_RECV_EVENT_TEXT** structure is sent to a client when they have requested to be notified of a message being sent using [SimConnect_Text](#).

Syntax

```
struct SIMCONNECT_RECV_EVENT_TEXT : public SIMCONNECT_RECV_EVENT {  
    SIMCONNECT_TEXT_TYPE eTextType;  
    SIMCONNECT_TEXT_ORIGIN eOrigin;  
    float fDuration;  
    DWORD dwFlags;  
    DWORD dwUnitSize;  
    BYTE rgMessage[1];  
};
```

Members

eTextType

Enum value, defined in [SIMCONNECT_TEXT_TYPE](#), containing the type of message.

eOrigin

Enum value, defined in [SIMCONNECT_TEXT_ORIGIN](#), containing the origin of the message.

fDuration

Float containing the duration that the message is visible.

dwFlags

Double word containing any flags. Reserved for future use.

dwUnitSize

Double word containing the length of the message. Used to iterate over menu-type messages.

rgMessage[1]

BYTE array containing the message. Menu-type messages are separated by a null terminator.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_EVENT_TEXT**.

See Also

- [SimConnect_SubscribeToSystemEvent](#)
-

SIMCONNECT_RECV_EVENT_TEXT_DESTROYED

The **SIMCONNECT_RECV_EVENT_TEXT_DESTROYED** structure is sent to a client when they have requested to be notified of a message being destroyed.

Syntax

```
struct SIMCONNECT_RECV_EVENT_TEXT_DESTROYED : public  
SIMCONNECT_RECV_EVENT {  
    SIMCONNECT_TEXT_TYPE eTextType;  
    SIMCONNECT_TEXT_RESULT eTextResult;
```

```
    DWORD dwFlags;  
};
```

Members

eTextType

Enum value, defined in [SIMCONNECT_TEXT_TYPE](#), containing the type of message.

eTextResult

Enum value, defined in [SIMCONNECT_TEXT_RESULT](#), containing the reason for the message being destroyed.

dwFlags

Double word containing any flags. Reserved for future use.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_EVENT_TEXT_DESTROYED**.

See Also

- [SimConnect_SubscribeToSystemEvent](#)
-

SIMCONNECT_RECV_EVENT_WEAPON

The **SIMCONNECT_RECV_EVENT_WEAPON** structure is sent to a client when they have requested to be notified of a detachable weapon being fired or detonated.

Syntax

```
struct SIMCONNECT_RECV_EVENT_WEAPON : public SIMCONNECT_RECV_EVENT {  
    DWORD dwRequestID;  
    char szWeaponTitle[MAX_PATH];  
    char szWeaponType[MAX_PATH];  
    DWORD dwObjectID;  
    DWORD dwAttackerID;  
    DWORD dwTargetID;  
    SIMCONNECT_DATA_LATLONALT llaPosition;  
    DWORD dwResult;  
    DWORD dwFlags;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

szWeaponTitle[MAX_PATH]

Null terminated string containing the weapon title.

szWeaponType[MAX_PATH]

Null terminated string containing the weapon type.

dwObjectID

Double word containing the object ID of the weapon.

dwAttackerID

Double word containing the object ID of the object that fired the weapon.

dwTargetID

Double word containing the object ID of the weapon's designed target.

llaPosition

[SIMCONNECT_DATA_LATLONALT](#) structure containing the object position.

dwResult

Result of the weapon event. Specifies if the event was a firing (0), an armed hit (1), or an unarmed hit

(2).

dwFlags

Double word containing any flags. Reserved for future use.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_EVENT_WEAPON**.

See Also

- [SIMCONNECT_RECV_EVENT_COUNTERMEASURE](#)
 - [SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_COUNTERMEASURE

The **SIMCONNECT_RECV_EVENT_COUNTERMEASURE** structure is sent to a client when they have requested to be notified of a deployed countermeasure.

Syntax

```
struct SIMCONNECT_RECV_EVENT_COUNTERMEASURE : public  
SIMCONNECT_RECV_EVENT {  
    DWORD dwRequestID;  
    char szCountermeasureName[MAX_PATH];  
    DWORD dwObjectID;  
    DWORD dwAttackerID;  
    SIMCONNECT_DATA_LATLONALT llaPosition;  
    DWORD dwFlags;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

szCountermeasureName[MAX_PATH]

Null terminated string containing the countermeasure name.

dwObjectID

Double word containing the object ID of the countermeasure.

dwAttackerID

Double word containing the object ID of the object that released the countermeasure.

llaPosition

[SIMCONNECT_DATA_LATLONALT](#) structure containing the object position.

dwFlags

Double word containing any flags. Reserved for future use.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_EVENT_COUNTERMEASURE**.

See Also

- [SIMCONNECT_RECV_EVENT_WEAPON](#)
 - [SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON

The **SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON** structure is sent to a client when they have requested to be notified of an object being damaged by a weapon.

Syntax

```
struct SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON : public  
SIMCONNECT_RECV_EVENT {  
    DWORD dwRequestID;  
    SIMCONNECT_DATA_OBJECT_DAMAGED_BY_WEAPON DamageData;  
    DWORD dwFlags;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

DamageData

[SIMCONNECT_DATA_OBJECT_DAMAGED_BY_WEAPON](#) structure containing the data of an object damaged by a weapon.

dwFlags

Double word containing any flags. Reserved for future use.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to **SIMCONNECT_RECV_ID_EVENT_OBJECT_DAMAGED_BY_WEAPON**.

See Also

- [SIMCONNECT_RECV_EVENT_WEAPON](#)
 - [SIMCONNECT_RECV_EVENT_COUNTERMEASURE](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_FLIGHT_SEGMENT

Received by the client after a successful call to [SimConnect_RequestFlightSegmentDataByGUID](#) or [SimConnect_RequestFlightSegmentDataByIndex](#) containing data about the requested flight segment.

Syntax

```
struct SIMCONNECT_RECV_FLIGHT_SEGMENT : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    GUID guidInstanceID;  
    GUID guidSegmentGoalID;  
    DWORD dwParameterCount;  
    DWORD dwTotalRangeCount;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

guidInstanceID

Guid Instance id of the flight segment.

guidSegmentGoalID

Guid Instance id of the flight segment's goal.

dwParameterCount

Double word containing the number of parameters that are being graded in this flight segment.

dwTotalRangeCount

Double word containing the total number of ranges combined for all parameters that are graded with this flight segment.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_FLIGHT_SEGMENT](#).

See Also

- [SimConnect_RequestFlightSegmentDataByGUID](#)
 - [SimConnect_RequestFlightSegmentDataByIndex](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_FLIGHT_SEGMENT_READY_FOR_GRADING

Received by the client with a FlightSegmentReadyForGrading notification. Contains instance ID of the Flight Segment.

Syntax

```
struct SIMCONNECT_RECV_FLIGHT_SEGMENT_READY_FOR_GRADING : public  
SIMCONNECT_RECV_EVENT_BASE {  
    GUID guidInstanceID;  
};
```

Members

guidInstanceID

Guid Instance id of the flight segment that is ready for grading.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_FLIGHT_SEGMENT_READY_FOR_GRADING](#).

See Also

- [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

-
- [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_GOAL

Received by the client after a successful call to [SimConnect_RequestGoalDataByGUID](#) or [SimConnect_RequestGoalDataByIndex](#).

Syntax

```
struct SIMCONNECT_RECV_GOAL : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    GUID guidInstanceID;  
    BOOL isOptional;  
    DWORD dwOrder;  
    DWORD dwPointValue;  
    SIMCONNECT_GOAL_STATE eGoalState;  
    DWORD dwChildGoalCount;  
    char szGoalText[256];  
    char szGoalSucceededText[256];  
    char szGoalFailedText[256];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

guidInstanceID

Guid Instance id of the goal.

isOptional

Boolean value, true if the goal is optional to scenario completion.

dwOrder

Double word containing the priority order of the goal.

dwPointValue

Double word containing the point value of the goal.

eGoalState

Enum value, defined in [SIMCONNECT_GOAL_STATE](#), specifying the state of the goal.

dwChildGoalCount

Double word specifying how many child goals are attached to this goal.

szGoalText

Null-terminated string containing the goal text.

szGoalSucceededText

Null-terminated string containing additional text to articulate the Goal Passed state.

szGoalFailedText

Null-terminated string containing additional text to articulate the Goal Failed state.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_GOAL](#).

See Also

- [SIMCONNECT_GOAL_STATE](#)
 - [SimConnect_RequestGoalDataByGUID](#)
 - [SimConnect_RequestGoalDataByIndex](#)
 - [SimConnect API Reference](#)
-

-
- [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_GOAL_PAIR

Received by the client after a successful call to [SimConnect_RequestChildGoalDataByIndex](#).

Syntax

```
struct SIMCONNECT_RECV_GOAL_PAIR : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    GUID guidChildInstanceID;  
    GUID guidParentInstanceID;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

guidChildInstanceID

Guid Instance id of the child goal.

guidParentInstanceID

Guid Instance id of the parent goal that owns the child.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_GOAL_PAIR](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_MISSION_OBJECT_COUNT

Received by the client after a successful call to [SimConnect_RequestFlightSegmentCount](#), [SimConnect_RequestGoalCount](#), or [SimConnect_RequestMissionObjectiveCount](#).

Syntax

```
struct SIMCONNECT_RECV_MISSION_OBJECT_COUNT : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    SIMCONNECT_MISSION_OBJECT_TYPE eMissionObjectType;  
    DWORD dwCount;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

eMissionObjectType

The requested object type whose value types reside in the [SIMCONNECT_MISSION_OBJECT_TYPE](#) enumeration.

dwCount

Double word containing the count of the requested object type in the active scenario.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_MISSION_OBJECT_COUNT](#).

See Also

- [SIMCONNECT_MISSION_OBJECT_TYPE](#)
 - [SimConnect_RequestFlightSegmentCount](#)
 - [SimConnect_RequestGoalCount](#)
 - [SimConnect_RequestMissionObjectiveCount](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_MISSION_OBJECTIVE

Received by the client after a successful call to
[SimConnect_RequestMissionObjectiveDataByGUID](#) or
[SimConnect_RequestMissionObjectiveDataByIndex](#).

Syntax

```
struct SIMCONNECT_RECV_MISSION_OBJECTIVE : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    GUID guidInstanceID;  
    BOOL isOptional;  
    BOOL isObjectivePassed;  
    SIMCONNECT_MISSION_OBJECTIVE_STATUS eMissionObjectiveStatus;  
    DWORD dwPassValue;  
    DWORD dwCurrentScore;  
    DWORD dwTotalPossiblePoints;  
    DWORD dwPointValue;  
    DWORD dwOrder;  
    DWORD dwChildGoalCount;  
    char szMissionObjectiveText[256];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

guidInstanceID

Guid Instance id of the mission objective.

isOptional

Boolean value, true if the mission objective is optional to scenario completion.

isObjectivePassed

Boolean value, true if the mission objective is passed.

eMissionObjectiveStatus

Enumeration value, defined as [SIMCONNECT_MISSION_OBJECTIVE_STATUS](#), that specifies the status of the mission objective .

dwPassValue

Double word containing the value needed to pass the mission objective.

dwCurrentScore

Double word containing the current score of the mission objective.

dwTotalPossiblePoints

Double word containing the maximum total score possible for the mission objective.

dwPointValue

Double word containing the client defined request ID.

dwOrder

Double word containing the number of points the mission objective is worth when passed.

dwChildGoalCount

Double word specifying how many child goals are attached to this mission objective.

szMissionObjectiveText

Null-terminated string containing the mission objective text.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_MISSION_OBJECTIVE**.

See Also

- [SIMCONNECT_MISSION_OBJECTIVE_STATUS](#)
 - [SimConnect_RequestMissionObjectiveDataByGUID](#)
 - [SimConnect_RequestMissionObjectiveDataByIndex](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_PARAMETER_RANGE

Received by the client after a successful call to [SimConnect_RequestFlightSegmentRangeData](#).

Syntax

```
struct SIMCONNECT_RECV_PARAMETER_RANGE : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwExceededCount;  
    DWORD dwMaxOverMeasured;  
    DWORD dwMinUnderMeasured;  
    GUID guidFlightSegmentID;  
    DWORD dwColorRedComponent;  
    DWORD dwColorGreenComponent;  
    DWORD dwColorBlueComponent;  
    char szRangeName[256];  
    char szParameterName[256];  
    char szParameterUnitsString[256];  
};
```

Members**dwRequestID**

Double word containing the client defined request ID.

dwExceededCount

Double word containing the number of times this range was exceeded in either direction, checking every second.

dwMaxOverMeasured

Double word containing the upper bound of this range, in terms of value above the measured value.

dwMinUnderMeasured

Double word containing the lower bound of this range, in terms of value below the measured value.

guidFlightSegmentID

The guid instance ID of the flight segment that owns this parameter range.

dwColorRedComponent

The Red component of the Range Color specified as a byte (0-255).

dwColorGreenComponent

The Green component of the Range Color specified as a byte (0-255).

dwColorBlueComponent

The Blue component of the Range Color specified as a byte (0-255).

szRangeName

Null-terminated string containing the range name.

szParameterName

Null-terminated string containing the parameter name.

szParameterUnitsString

Null-terminated string containing the units that the parameter is in terms of.

Working Samples

Primary samples [Managed Mission Objects](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_PARAMETER_RANGE](#).

See Also

- [SimConnect_RequestFlightSegmentRangeData](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_PLAYBACK_STATE_CHANGED

Received by the client with a PlaybackStateChanged notification. Contains whether playback has started or ended and the file name of the recording.

Syntax

```
struct SIMCONNECT_RECV_PLAYBACK_STATE_CHANGED : public  
SIMCONNECT_RECV_EVENT {  
    BOOL hasPlaybackStarted;  
    char szRecordingFileName[256];  
};
```

Members**hasPlaybackStarted**

If true, playback has started. If false, playback has completed.

szRecordingFileName

Null-terminated string containing the file name of the recording that is being played or just finished playing.

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_PLAYBACK_STATE_CHANGED](#).

See Also

- [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
-

-
- [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_RECORDER_STATE_CHANGED

Received by the client with a RecorderStateChanged notification. Contains whether recording has started or ended.

Syntax

```
struct SIMCONNECT_RECV_RECORDER_STATE_CHANGED : public  
SIMCONNECT_RECV_EVENT {  
    BOOL hasRecordingStarted;  
};
```

Members

hasRecordingStarted

If true, recording has started. If false, recording has completed.

Remarks

This structure inherits the [SIMCONNECT_RECV_EVENT](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_RECORDER_STATE_CHANGED](#).

See Also

- [SimConnect_SubscribeToSystemEvent](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_RECORDING_INFO

Received by the client after a successful call to [SimConnect_RequestRecordingInfo](#)

Syntax

```
struct SIMCONNECT_RECV_RECORDING_INFO : public SIMCONNECT_RECV_EVENT {  
    DWORD dwRequestID;  
    double startTime;  
    double endTime;  
    DWORD bookmarkCount;  
    char szTitle[MAX_PATH];  
    char szDescription[MAX_PATH];  
    char szUserContainerTitle[MAX_PATH];  
    char szFilename[MAX_PATH];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

startTime

The start time of the recording in absolute seconds in simulation zulu time.

endTime

The end time of the recording in absolute seconds in simulation zulu time.

bookmarkCount

The number of bookmarks in the recording.

szTitle

The title of the recording.

szDescription

The description of the recording.

szUserContainerTitle

The container title of the user object used during the recording.

szFilename

The file name of the recording.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_RECORDING_INFO](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_RECORDING_INFO_W

Received by the client after a successful call to [SimConnect_RequestRecordingInfoW](#)

Syntax

```
struct SIMCONNECT_RECV_RECORDING_INFO_W : public SIMCONNECT_RECV_EVENT
{
    DWORD dwRequestID;
    double startTime;
    double endTime;
    DWORD bookmarkCount;
    wchar_t szTitle[MAX_PATH];
    wchar_t szDescription[MAX_PATH];
    wchar_t szUserContainerTitle[MAX_PATH];
    wchar_t szFilename[MAX_PATH];
};
```

Members

dwRequestID

Double word containing the client defined request ID.

startTime

The start time of the recording in absolute seconds in simulation zulu time.

endTime

The end time of the recording in absolute seconds in simulation zulu time.

bookmarkCount

The number of bookmarks in the recording.

szTitle

The title of the recording.

szDescription

The description of the recording.

szUserContainerTitle

The container title of the user object used during the recording.

szFilename

The file name of the recording.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_RECORDING_INFO_W**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_RECORDING_BOOKMARK_INFO

Received by the client after a successful call to [SimConnect_RequestBookmarkInfo](#)

Syntax

```
struct SIMCONNECT_RECV_RECORDING_BOOKMARK_INFO : public  
SIMCONNECT_RECV_EVENT {  
    DWORD dwRequestID;  
    double timeStamp;  
    DWORD bookmarkIndex;  
    char szTitle[MAX_PATH];  
    char szRecordingFilename[MAX_PATH];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

timeStamp

The time of the bookmark in absolute seconds in simulation zulu time.

bookmarkIndex

The index of the bookmark.

szTitle

The title of the bookmark.

szRecordingFilename

The file name of the recording for the bookmark.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_RECORDING_BOOKMARK_INFO**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_RECORDING_BOOKMARK_INFO_W

Received by the client after a successful call to [SimConnect_RequestBookmarkInfoW](#)

Syntax

```
struct SIMCONNECT_RECV_RECORDING_BOOKMARK_INFO_W : public  
SIMCONNECT_RECV_EVENT {  
    DWORD dwRequestID;  
    double timeStamp;  
    DWORD bookmarkIndex;  
    wchar_t szTitle[MAX_PATH];  
    wchar_t szRecordingFilename[MAX_PATH];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

timeStamp

The time of the bookmark in absolute seconds in simulation zulu time.

bookmarkIndex

The index of the bookmark.

szTitle

The title of the bookmark.

szRecordingFilename

The file name of the recording for the bookmark.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_RECORDING_BOOKMARK_INFO_W](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_LANDING_TRIGGER_INFO

Received by the client after a successful call to [SimConnect_RequestLandingTriggerLandingInfoCount](#).

Syntax

```
struct SIMCONNECT_RECV_LANDING_TRIGGER_INFO : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    GUID landingTriggerInstanceID;  
    int landingsCount;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

landingTriggerInstanceID

Guid Instance id of the landing trigger which can be used in [SimConnect_RequestLandingTriggerLandingInfoByIndex](#).

landingsCount

The total amount of landings that met the landing trigger's criteria.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_LANDING_TRIGGER_INFO**.

See Also

- [SimConnect_RequestLandingTriggerLandingInfoCount](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_LANDING_INFO

Received by the client after a successful call to

[SimConnect_RequestLandingTriggerLandingInfoByIndex](#).

Syntax

```
struct SIMCONNECT_RECV_LANDING_INFO : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    GUID LandingTriggerInstanceID;  
    SIMCONNECT_LANDING_TYPE LandingType;  
    double Latitude;  
    double Longitude;  
    double Altitude;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

landingTriggerInstanceID

Guid Instance id of the landing trigger.

LandingType

Type of landing trigger specified by the [SIMCONNECT_LANDING_TYPE](#) enumeration.

Latitude

Latitude of the specified landing in degrees.

Longitude

Longitude of the specified landing in degrees.

Altitude

Altitude of the specified landing in feet.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_LANDING_INFO**.

See Also

- [SimConnect_RequestLandingTriggerLandingInfoByIndex](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_GROUND_INFO

The **SIMCONNECT_RECV_GROUND_INFO** structure is used to return data in response to a call to [SimConnect_RequestGroundInfo](#) or [SimConnect_RequestGroundInfoOnSimObject](#).

Syntax

```
struct SIMCONNECT_RECV_GROUND_INFO : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwGridWidth;  
    DWORD dwGridHeight;  
    DWORD dwArraySize;  
    DWORD dwFlags;  
    BOOL bNotAllValid;  
    SIMCONNECT_DATA_GROUND_INFO rgData[1];  
};
```

Members

dwRequestID;

Double word containing the client defined request ID.

dwGridWidth;

Double word containing the width of the requested grid.

dwGridHeight;

Double word containing the height of the requested grid.

dwArraySize;

Double word containing the number of items in the **rgData** array (GridWidth * GridHeight).

dwFlags;

Double word containing the flags that were passed to the [SimConnect_RequestGroundInfo](#) or [SimConnect_RequestGroundInfoOnSimObject](#) function.

bNotAllValid;

Boolean value, true if any of the returned data points is invalid
(**SIMCONNECT_DATA_GROUND_INFO.bIsValid** is false)

rgData[1];

First element in an array of [SIMCONNECT_DATA_GROUND_INFO](#) items, **dwArraySize** long.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_GROUND_INFO**.

This structure returns an array of ground info points in response to a call to

[SimConnect_RequestGroundInfo](#) or [SimConnect_RequestGroundInfoOnSimObject](#).

See the remarks for [SimConnect_RequestGroundInfo](#) for more information.

See Also

- [SIMCONNECT_DATA_GROUND_INFO](#)
 - [SimConnect_RequestGroundInfo](#)
 - [SimConnect_RequestGroundInfoOnSimObject](#)
 - [SimConnect API Reference](#)
-

SIMCONNECT_RECV_FACILITIES_LIST

The **SIMCONNECT_RECV_FACILITIES_LIST** structure is used to provide information on the number of elements in a list of facilities returned to the client, and the number of packets that were used to transmit the data.

Syntax

```
struct SIMCONNECT_RECV_FACILITIES_LIST : public SIMCONNECT_RECV{
```

```
    DWORD dwRequestID;
    DWORD dwArraySize;
    DWORD dwEntry;
    DWORD dwOutOf;
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwArraySize

Double word containing the number of elements in the list that are within this packet. For example, if there are 25 airports returned in the [SIMCONNECT_RECV_AIRPORT_LIST](#) structure, then this field will contain 25, but if there are 400 airports in the list and the data is returned in two packets, then this value will contain the number of entries within each packet.

dwEntry

Double word containing the index number of this list packet. This number will be from 0 to *dwOutOf* - 1.

dwOutOf

Double word containing the total number of packets used to transmit the list.

Remarks

This structure is inherited by [SIMCONNECT_RECV_AIRPORT_LIST](#), [SIMCONNECT_RECV_NDB_LIST](#), [SIMCONNECT_RECV_VOR_LIST](#), [SIMCONNECT_RECV_TACAN_LIST](#) and [SIMCONNECT_RECV_WAYPOINT_LIST](#).

This structure inherits the [SIMCONNECT_RECV](#) structure, so use the [SIMCONNECT_RECV_ID](#) enumeration to determine which list structure has been received.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect_RequestFacilitiesList](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO

The **SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO** structure is used to provide information on the currently connected joystick devices.

Syntax

```
struct SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO : public SIMCONNECT_RECV{
    DWORD dwRequestID;
    DWORD dwArraySize;
    SIMCONNECT_DATA_JOYSTICK_DEVICE_INFO rgData[1];
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwArraySize

Double word containing the number of joystick devices in the list that are within this packet.

rgData

Array of [SIMCONNECT_DATA_JOYSTICK_DEVICE_INFO](#) structures.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure, so use the [SIMCONNECT_RECV_ID](#) enumeration to determine which list structure has been received.

See the remarks for [SimConnect_RequestJoystickDeviceInfo](#).

See Also

- [SimConnect_RequestJoystickDeviceInfo](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS

The **SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS** structure is used to provide identification information on mobile scenery objects within a specified radius of the user vehicle.

Syntax

```
struct SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS : public  
SIMCONNECT_RECV{  
    DWORD dwRequestID;  
    DWORD dwArraySize;  
    SIMCONNECT_DATA_MOBILE_SCENERY_INFO rgData[1];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwArraySize

Double word containing the number of mobile scenery objects in the list that are within this packet.

rgData

Array of [SIMCONNECT_DATA_MOBILE_SCENERY_INFO](#) structures.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure, so use the [SIMCONNECT_RECV_ID](#) enumeration to determine which list structure has been received.

See the remarks for [SimConnect_RequestMobileSceneryInRadius](#).

See Also

- [SimConnect_RequestMobileSceneryInRadius](#)
 - [SimConnect_RequestMobileSceneryDataByID](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_DATA_MOBILE_SCENERY_INFO](#)
 - [SIMCONNECT_RECV_MOBILE_SCENERY_DATA](#)
-

SIMCONNECT_RECV_MOBILE_SCENERY_DATA

The **SIMCONNECT_RECV_MOBILE_SCENERY_DATA** structure is used to provide data about a mobile scenery object with the specified object ID.

Syntax

```
struct SIMCONNECT_RECV_MOBILE_SCENERY_DATA : public SIMCONNECT_RECV{
```

```
DWORD dwRequestID;
SIMCONNECT_GUID guidInstanceID;
SIMCONNECT_DATA_LATLONALT llaWorldPosition;
SIMCONNECT_DATA_PBH pbhWorldRotation;
BOOL isOnGround;
double speedKnots;
float scale;
DWORD dwObjectID;
};
```

Members

dwRequestID

Double word containing the client defined request ID.

guidInstanceID

Scenario Instance id of the mobile scenery object.

llaWorldPosition

A [SIMCONNECT_DATA_LATLONALT](#) containing the world position of the mobile scenery object in degrees (altitude in feet).

pbhWorldRotation

A [SIMCONNECT_DATA_PBH](#) containing the world rotation of the mobile scenery object in radians.

isOnGround

A bool containing whether or not the mobile scenery object is clamped to the ground.

speedKnots

A double containing the speed of the mobile scenery object in knots.

scale

A float containing the scale of the mobile scenery object. 1 is the default scale of the object. Anything else is the default size of the object multiplied by the scale.

dwObjectID

Double word containing the object ID of the mobile scenery object.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure, so use the [SIMCONNECT_RECV_ID](#) enumeration to determine which list structure has been received.

See the remarks for [SimConnect_RequestMobileSceneryDataByID](#).

See Also

- [SimConnect_RequestMobileSceneryInRadius](#)
 - [SimConnect_RequestMobileSceneryDataByID](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_DATA_MOBILE_SCENERY_INFO](#)
 - [SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS](#)
-

SIMCONNECT_RECV_ID

The **SIMCONNECT_RECV_ID** enumeration type is used within the [SIMCONNECT_RECV](#) structure to indicate which type of structure has been returned.

Syntax

```
enum SIMCONNECT_RECV_ID{
    SIMCONNECT_RECV_ID_NULL,
    SIMCONNECT_RECV_ID_EXCEPTION,
    SIMCONNECT_RECV_ID_OPEN,
    SIMCONNECT_RECV_ID_QUIT,
    SIMCONNECT_RECV_ID_EVENT,
```

```
SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE,
SIMCONNECT_RECV_ID_EVENT_FILENAME,
SIMCONNECT_RECV_ID_EVENT_FRAME,
SIMCONNECT_RECV_ID_SIMOBJECT_DATA,
SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE,
SIMCONNECT_RECV_ID_WEATHER_OBSERVATION,
SIMCONNECT_RECV_ID_CLOUD_STATE,
SIMCONNECT_RECV_ID_ASSIGNED_OJBECT_ID,
SIMCONNECT_RECV_ID_RESERVED_KEY,
SIMCONNECT_RECV_ID_CUSTOM_ACTION,
SIMCONNECT_RECV_ID_SYSTEM_STATE,
SIMCONNECT_RECV_ID_CLIENT_DATA,
SIMCONNECT_RECV_ID_EVENT_WEATHER_MODE,
SIMCONNECT_RECV_ID_AIRPORT_LIST,
SIMCONNECT_RECV_ID_VOR_LIST,
SIMCONNECT_RECV_ID_NDB_LIST,
SIMCONNECT_RECV_ID_TACAN_LIST,
SIMCONNECT_RECV_ID_WAYPOINT_LIST,
SIMCONNECT_RECV_ID_EVENT_MULTIPLAYER_SERVER_STARTED,
SIMCONNECT_RECV_ID_EVENT_MULTIPLAYER_CLIENT_STARTED,
SIMCONNECT_RECV_ID_EVENT_MULTIPLAYER_SESSION_ENDED,
SIMCONNECT_RECV_ID_EVENT_RACE_END,
SIMCONNECT_RECV_ID_EVENT_RACE_LAP,
SIMCONNECT_RECV_ID_OBSERVER_DATA,
SIMCONNECT_RECV_ID_GROUND_INFO,
SIMCONNECT_RECV_ID_SYNCHRONOUS_BLOCK,
SIMCONNECT_RECV_ID_EXTERNAL_SIM_CREATE,
SIMCONNECT_RECV_ID_EXTERNAL_SIM_DESTROY,
SIMCONNECT_RECV_ID_EXTERNAL_SIM_SIMULATE,
SIMCONNECT_RECV_ID_EXTERNAL_SIM_LOCATION_CHANGED,
SIMCONNECT_RECV_ID_EXTERNAL_SIM_EVENT,
SIMCONNECT_RECV_ID_EVENT_WEAPON,
SIMCONNECT_RECV_ID_EVENT_COUNTERMEASURE,
SIMCONNECT_RECV_ID_EVENT_OBJECT_DAMAGED_BY_WEAPON,
SIMCONNECT_RECV_ID_VERSION,
SIMCONNECT_RECV_ID_SCENERY_COMPLEXITY,
SIMCONNECT_RECV_ID_SHADOW_FLAGS,
SIMCONNECT_RECV_ID_TACAN_LIST,
SIMCONNECT_RECV_ID_CAMERA_6DOF,
SIMCONNECT_RECV_ID_CAMERA_FOV,
SIMCONNECT_RECV_ID_CAMERA_SENSOR_MODE,
SIMCONNECT_RECV_ID_CAMERA_WINDOW_POSITION,
SIMCONNECT_RECV_ID_CAMERA_WINDOW_SIZE,
SIMCONNECT_RECV_ID_MISSION_OBJECT_COUNT,
SIMCONNECT_RECV_ID_GOAL,
SIMCONNECT_RECV_ID_MISSION_OBJECTIVE,
SIMCONNECT_RECV_ID_FLIGHT_SEGMENT,
SIMCONNECT_RECV_ID_PARAMETER_RANGE,
SIMCONNECT_RECV_ID_FLIGHT_SEGMENT_READY_FOR_GRADING,
SIMCONNECT_RECV_ID_GOAL_PAIR,
SIMCONNECT_RECV_ID_EVENT_FLIGHT_ANALYSIS_DIAGRAMS,
SIMCONNECT_RECV_ID_LANDING_TRIGGER_INFO,
SIMCONNECT_RECV_ID_LANDING_INFO,
SIMCONNECT_RECV_ID_SESSION_DURATION,
SIMCONNECT_RECV_ID_ATTACHPOINT_DATA,
SIMCONNECT_RECV_ID_PLAYBACK_STATE_CHANGED,
SIMCONNECT_RECV_ID_RECORDER_STATE_CHANGED,
SIMCONNECT_RECV_ID_RECORDING_INFO,
SIMCONNECT_RECV_ID_RECORDING_BOOKMARK_INFO,
SIMCONNECT_RECV_ID_TRAFFIC_SETTINGS,
SIMCONNECT_RECV_ID_JOYSTICK_DEVICE_INFO,
SIMCONNECT_RECV_ID_MOBILE_SCENERY_IN_RADIUS,
SIMCONNECT_RECV_ID_MOBILE_SCENERY_DATA,
```

```
SIMCONNECT_RECV_ID_EVENT_64,  
SIMCONNECT_RECV_ID_EVENT_TEXT,  
SIMCONNECT_RECV_ID_RECORDING_INFO_W,  
SIMCONNECT_RECV_ID_RECORDING_BOOKMARK_INFO_W,  
SIMCONNECT_RECV_ID_SYSTEM_STATE_W,  
SIMCONNECT_RECV_ID_EVENT_FILENAME_W,  
};
```

Members

SIMCONNECT_RECV_ID_NULL

Specifies that nothing useful has been returned.

SIMCONNECT_RECV_ID_EXCEPTION

Specifies that a [SIMCONNECT_RECV_EXCEPTION](#) structure has been received.

SIMCONNECT_RECV_ID_OPEN

Specifies that a [SIMCONNECT_RECV_OPEN](#) structure has been received.

SIMCONNECT_RECV_ID_QUIT

Specifies that the user has exited from *Prepar3D*.

SIMCONNECT_RECV_ID_EVENT

Specifies that a [SIMCONNECT_RECV_EVENT](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE

Specifies that a [SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_FILENAME

Specifies that a [SIMCONNECT_RECV_EVENT_FILENAME](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_FRAME

Specifies that a [SIMCONNECT_RECV_EVENT_FRAME](#) structure has been received.

SIMCONNECT_RECV_ID_SIMOBJECT_DATA

Specifies that a [SIMCONNECT_RECV_SIMOBJECT_DATA](#) structure has been received.

SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE

Specifies that a [SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE](#) structure has been received.

SIMCONNECT_RECV_ID_WEATHER_OBSERVATION

Specifies that a [SIMCONNECT_RECV_WEATHER_OBSERVATION](#) structure has been received.

SIMCONNECT_RECV_ID_CLOUD_STATE

Specifies that a [SIMCONNECT_RECV_CLOUD_STATE](#) structure has been received.

SIMCONNECT_RECV_ID_ASSIGNED_OBJECT_ID

Specifies that a [SIMCONNECT_RECV_ASSIGNED_OBJECT_ID](#) structure has been received.

SIMCONNECT_RECV_ID_RESERVED_KEY

Specifies that a [SIMCONNECT_RECV_RESERVED_KEY](#) structure has been received.

SIMCONNECT_RECV_ID_CUSTOM_ACTION

Specifies that a [SIMCONNECT_RECV_CUSTOM_ACTION](#) structure has been received.

SIMCONNECT_RECV_ID_SYSTEM_STATE

Specifies that a [SIMCONNECT_RECV_SYSTEM_STATE](#) structure has been received.

SIMCONNECT_RECV_ID_CLIENT_DATA

Specifies that a [SIMCONNECT_RECV_CLIENT_DATA](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_WEATHER_MODE

Specifies that the *dwData* parameter will contain one value of the [SIMCONNECT_WEATHER_MODE](#) enumeration.

SIMCONNECT_RECV_ID_AIRPORT_LIST

Specifies that a [SIMCONNECT_RECV_AIRPORT_LIST](#) structure has been received.

SIMCONNECT_RECV_ID_VOR_LIST

Specifies that a [SIMCONNECT_RECV_VOR_LIST](#) structure has been received.

SIMCONNECT_RECV_ID_NDB_LIST

Specifies that a [SIMCONNECT_RECV_NDB_LIST](#) structure has been received.

SIMCONNECT_RECV_ID_TACAN_LIST

Specifies that a [SIMCONNECT_RECV_TACAN_LIST](#) structure has been received.

SIMCONNECT_RECV_ID_WAYPOINT_LIST

Specifies that a [SIMCONNECT_RECV_WAYPOINT_LIST](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_MULTIPLAYER_SERVER_STARTED

Specifies that a [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SERVER_STARTED](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_MULTIPLAYER_CLIENT_STARTED

Specifies that a [SIMCONNECT_RECV_EVENT_MULTIPLAYER_CLIENT_STARTED](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_MULTIPLAYER_SESSION_ENDED

Specifies that a [SIMCONNECT_RECV_EVENT_MULTIPLAYER_SESSION_ENDED](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_RACE_END

Specifies that a [SIMCONNECT_RECV_EVENT_RACE_END](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_RACE_LAP

Specifies that a [SIMCONNECT_RECV_EVENT_RACE_LAP](#) structure has been received.

SIMCONNECT_RECV_ID_OBSERVER_DATA

Specifies that a [SIMCONNECT_RECV_OBSERVER_DATA](#) structure has been received.

SIMCONNECT_RECV_ID_GROUND_INFO

Specifies that a [SIMCONNECT_RECV_GROUND_INFO](#) structure has been received.

SIMCONNECT_RECV_ID_SYNCHRONOUS_BLOCK

Specifies that a [SIMCONNECT_RECV_SYNCHRONOUS_BLOCK](#) structure has been received.

SIMCONNECT_RECV_ID_EXTERNAL_SIM_CREATE

Specifies that a [SIMCONNECT_RECV_EXTERNAL_SIM_CREATE](#) structure has been received.

SIMCONNECT_RECV_ID_EXTERNAL_SIM_DESTROY

Specifies that a [SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY](#) structure has been received.

SIMCONNECT_RECV_ID_EXTERNAL_SIM_SIMULATE

Specifies that a [SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE](#) structure has been received.

SIMCONNECT_RECV_ID_EXTERNAL_SIM_LOCATION_CHANGED

Specifies that a [SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED](#) structure has been received.

SIMCONNECT_RECV_ID_EXTERNAL_SIM_EVENT

Specifies that a [SIMCONNECT_RECV_EXTERNAL_SIM_EVENT](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_WEAPON

Specifies that a [SIMCONNECT_RECV_EVENT_WEAPON](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_COUNTERMEASURE

Specifies that a [SIMCONNECT_RECV_EVENT_COUNTERMEASURE](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_OBJECT_DAMAGED_BY_WEAPON

Specifies that a

[SIMCONNECT_RECV_ID_EVENT_OBJECT_DAMAGED_BY_WEAPON](#) structure has been received.

SIMCONNECT_RECV_ID_VERSION

Specifies that a [SIMCONNECT_RECV_VERSION](#) structure has been received.

SIMCONNECT_RECV_ID_SCENERY_COMPLEXITY

Specifies that a [SIMCONNECT_RECV_SCENERY_COMPLEXITY](#) structure has been received.

SIMCONNECT_RECV_ID_SHADOW_FLAGS

Specifies that a [**SIMCONNECT_RECV_SHADOW_FLAGS**](#) structure has been received.

SIMCONNECT_RECV_ID_SESSION_DURATION

Specifies that a [**SIMCONNECT_RECV_SESSION_DURATION**](#) structure has been received.

SIMCONNECT_RECV_ID_TACAN_LIST

Specifies that a [**SIMCONNECT_RECV_TACAN_LIST**](#) structure has been received.

SIMCONNECT_RECV_ID_CAMERA_6DOF

Specifies that a [**SIMCONNECT_RECV_CAMERA_6DOF**](#) structure has been received.

SIMCONNECT_RECV_ID_CAMERA_FOV

Specifies that a [**SIMCONNECT_RECV_CAMERA_FOV**](#) structure has been received.

SIMCONNECT_RECV_ID_CAMERA_SENSOR_MODE

Specifies that a [**SIMCONNECT_RECV_CAMERA_SENSOR_MODE**](#) structure has been received.

SIMCONNECT_RECV_ID_CAMERA_WINDOW_POSITION

Specifies that a [**SIMCONNECT_RECV_CAMERA_WINDOW_POSITION**](#) structure has been received.

SIMCONNECT_RECV_ID_CAMERA_WINDOW_SIZE

Specifies that a [**SIMCONNECT_RECV_CAMERA_WINDOW_SIZE**](#) structure has been received.

SIMCONNECT_RECV_ID_MISSION_OBJECT_COUNT

Specifies that a [**SIMCONNECT_RECV_MISSION_OBJECT_COUNT**](#) structure has been received.

SIMCONNECT_RECV_ID_GOAL

Specifies that a [**SIMCONNECT_RECV_GOAL**](#) structure has been received.

SIMCONNECT_RECV_ID_MISSION_OBJECTIVE

Specifies that a [**SIMCONNECT_RECV_MISSION_OBJECTIVE**](#) structure has been received.

SIMCONNECT_RECV_ID_FLIGHT_SEGMENT

Specifies that a [**SIMCONNECT_RECV_FLIGHT_SEGMENT**](#) structure has been received.

SIMCONNECT_RECV_ID_PARAMETER_RANGE

Specifies that a [**SIMCONNECT_RECV_PARAMETER_RANGE**](#) structure has been received.

SIMCONNECT_RECV_ID_FLIGHT_SEGMENT_READY_FOR_GRADING

Specifies that a [**SIMCONNECT_RECV_FLIGHT_SEGMENT_READY_FOR_GRADING**](#) structure has been received.

SIMCONNECT_RECV_ID_GOAL_PAIR

Specifies that a [**SIMCONNECT_RECV_GOAL_PAIR**](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_FLIGHT_ANALYSIS_DIAGRAMS

Specifies that a [SIMCONNECT_RECV_EVENT_FLIGHT_ANALYSIS_DIAGRAMS](#) structure has been received.

SIMCONNECT_RECV_ID_LANDING_TRIGGER_INFO

Specifies that a [SIMCONNECT_RECV_LANDING_TRIGGER_INFO](#) structure has been received.

SIMCONNECT_RECV_ID_LANDING_INFO

Specifies that a [SIMCONNECT_RECV_LANDING_INFO](#) structure has been received.

SIMCONNECT_RECV_ID_SESSION_DURATION

Specifies that a [SIMCONNECT_RECV_SESSION_DURATION](#) structure has been received.

SIMCONNECT_RECV_ID_ATTACHPOINT_DATA

Specifies that a [SIMCONNECT_RECV_ATTACHPOINT_DATA](#) structure has been received.

SIMCONNECT_RECV_ID_PLAYBACK_STATE_CHANGED

Specifies that a [SIMCONNECT_RECV_PLAYBACK_STATE_CHANGED](#) structure has been received.

SIMCONNECT_RECV_ID_RECORDER_STATE_CHANGED

Specifies that a [SIMCONNECT_RECV_RECORDER_STATE_CHANGED](#) structure has been received.

SIMCONNECT_RECV_ID_RECORDING_INFO

Specifies that a [SIMCONNECT_RECV_RECORDING_INFO](#) structure has been received.

SIMCONNECT_RECV_ID_RECORDING_BOOKMARK_INFO

Specifies that a [SIMCONNECT_RECV_RECORDING_BOOKMARK_INFO](#) structure has been received.

SIMCONNECT_RECV_ID_TRAFFIC_SETTINGS

Specifies that a [SIMCONNECT_RECV_TRAFFIC_SETTINGS](#) structure has been received.

SIMCONNECT_RECV_ID_JOYSTICK_DEVICE_INFO

Specifies that a [SIMCONNECT_RECV_JOYSTICK_DEVICE_INFO](#) structure has been received.

SIMCONNECT_RECV_ID_MOBILE_SCENERY_IN_RADIUS

Specifies that a [SIMCONNECT_RECV_MOBILE_SCENERY_IN_RADIUS](#) structure has been received.

SIMCONNECT_RECV_ID_MOBILE_SCENERY_DATA

Specifies that a [SIMCONNECT_RECV_MOBILE_SCENERY_DATA](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_64

Specifies that a [SIMCONNECT_RECV_EVENT_64](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_TEXT

Specifies that a [SIMCONNECT_RECV_EVENT_TEXT](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_TEXT_DESTROYED

Specifies that a [SIMCONNECT_RECV_EVENT_TEXT_DESTROYED](#) structure has been received.

SIMCONNECT_RECV_ID_RECORDING_INFO_W

Specifies that a [SIMCONNECT_RECV_RECORDING_INFO_W](#) structure has been received.

SIMCONNECT_RECV_ID_RECORDING_BOOKMARK_INFO_W

Specifies that a [SIMCONNECT_RECV_RECORDING_BOOKMARK_INFO_W](#) structure has been received.

SIMCONNECT_RECV_ID_SYSTEM_STATE_W

Specifies that a [SIMCONNECT_RECV_SYSTEM_STATE_W](#) structure has been received.

SIMCONNECT_RECV_ID_EVENT_FILENAME_W

Specifies that a [SIMCONNECT_RECV_EVENT_FILENAME_W](#) structure has been received.

SIMCONNECT_RECV_NDB_LIST

The **SIMCONNECT_RECV_NDB_LIST** structure is used to return a list of [SIMCONNECT_DATA_FACILITY_NDB](#) structures.

Syntax

```
struct SIMCONNECT_RECV_NDB_LIST : public SIMCONNECT_RECV_FACILITIES_LIST{
    SIMCONNECT_DATA_FACILITY_NDB rgData[1];
};
```

Members

rgData[1]

Array of [SIMCONNECT_DATA_FACILITY_NDB](#) structures.

Remarks

This structure inherits the [SIMCONNECT_RECV_FACILITIES_LIST](#) structure, which identifies the number of elements in the list, and the number of packets needed to transmit all the data.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect_RequestFacilitiesList](#)
- [SIMCONNECT_RECV_AIRPORT_LIST](#)
- [SIMCONNECT_RECV_FACILITIES_LIST](#)
- [SIMCONNECT_RECV_VOR_LIST](#)
- [SIMCONNECT_RECV_WAYPOINT_LIST](#)
- [SimConnect API Reference](#)

-
- [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_SIMOBJECT_DATA

The **SIMCONNECT_RECV_SIMOBJECT_DATA** structure will be received by the client after a successful call to [SimConnect_RequestDataOnSimObject](#) or [SimConnect_RequestDataOnSimObjectType](#).

Syntax

```
struct SIMCONNECT_RECV_SIMOBJECT_DATA : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwObjectID;  
    DWORD dwDefineID;  
    DWORD dwFlags;  
    DWORD dwentrynumber;  
    DWORD dwoutof;  
    DWORD dwDefineCount;  
    DWORD dwData;  
};
```

Members

dwRequestID

The ID of the client defined request.

dwObjectID

Double word containing the server defined object ID.

dwDefineID

The ID of the client defined data definition.

dwFlags

The flags that were set for this data request, see [SimConnect_RequestDataOnSimObject](#) for a description of the flags. This parameter will always be set to zero if the call was [SimConnect_RequestDataOnSimObjectType](#).

dwentrynumber

If multiple objects are being returned, this is the index number of this object out of a total of **dwoutof**. This will always be 1 if the call was [SimConnect_RequestDataOnSimObject](#), and can be 0 or more if the call was [SimConnect_RequestDataOnSimObjectType](#).

dwoutof

The total number of objects being returned. Note that **dwentrynumber** and **dwoutof** start with 1 not 0, so if two objects are being returned **dwentrynumber** and **dwoutof** pairs will be 1,2 and 2,2 for the two objects. This will always be 1 if the call was [SimConnect_RequestDataOnSimObject](#), and can be 0 or more if the call was [SimConnect_RequestDataOnSimObjectType](#).

dwDefineCount

The number of 8-byte elements in the **dwData** array.

dwData

A data array containing information on a specified object in 8-byte (double word) elements. The length of the array is **dwDefineCount**.

Working Samples

[Request Data](#)

[Set Data](#)

Primary samples [Tagged Data](#)

[Throttle Control](#)

[Variable Strings](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter

of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_SIMOBJECT_DATA**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EVENT](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE](#)
-

SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE

The **SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE** structure will be received by the client after a successful call to [SimConnect_RequestDataOnSimObjectType](#). It is an identical structure to [SIMCONNECT_RECV_SIMOBJECT_DATA](#).

Syntax

```
struct SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE : public  
SIMCONNECT_RECV_SIMOBJECT_DATA{  
};
```

Remarks

This structure inherits the [SIMCONNECT_RECV_SIMOBJECT_DATA](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_RECV_SIMOBJECT_DATA](#)
 - [SimConnect_RequestDataOnSimObjectType](#)
-

SIMCONNECT_RECV_OPEN

The **SIMCONNECT_RECV_OPEN** structure is used to return information to the client, after a successful call to [SimConnect_Open](#).

Syntax

```
struct SIMCONNECT_RECV_OPEN : public SIMCONNECT_RECV {  
    char szApplicationName[256];  
    DWORD dwApplicationVersionMajor;  
    DWORD dwApplicationVersionMinor;  
    DWORD dwApplicationBuildMajor;  
    DWORD dwApplicationBuildMinor;  
    DWORD dwSimConnectVersionMajor;  
    DWORD dwSimConnectVersionMinor;  
    DWORD dwSimConnectBuildMajor;  
    DWORD dwSimConnectBuildMinor;  
    DWORD dwReserved1;  
    DWORD dwReserved2;  
};
```

Members

szApplicationName[256]

Null-terminated string containing the application name.

dwApplicationVersionMajor

Double word containing the application version major number.

dwApplicationVersionMinor

Double word containing the application version minor number.

dwApplicationBuildMajor

Double word containing the application build major number.

dwApplicationBuildMinor

Double word containing the application build minor number.

dwSimConnectVersionMajor

Double word containing the SimConnect version major number.

dwSimConnectVersionMinor

Double word containing the SimConnect version minor number.

dwSimConnectBuildMajor

Double word containing the SimConnect build major number.

dwSimConnectBuildMinor

Double word containing the SimConnect build minor number.

dwReserved1

Reserved.

dwReserved2

Reserved.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_OPEN](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_RESERVED_KEY

The **SIMCONNECT_RECV_RESERVED_KEY** structure is used with the [SimConnect_RequestReservedKey](#) function to return the reserved key combination.

Syntax

```
struct SIMCONNECT_RECV_RESERVED_KEY : public SIMCONNECT_RECV {  
    char szChoiceReserved[30];  
    char szReservedKey[50];  
};
```

Members**szChoiceReserved[30]**

Null-terminated string containing the key that has been reserved. This will be identical to the string entered as one of the choices for the [SimConnect_RequestReservedKey](#) function.

szReservedKey[50]

Null-terminated string containing the reserved key combination. This will be an uppercase string containing all the modifiers that apply. For example, if the client program requests "q", and the choice is accepted, then this parameter will contain "TAB+Q". If the client program requests "Q", then this parameter will contain "SHIFT+TAB+Q". This string could then appear, for example, in a dialog from the client application, informing a user of the appropriate help key.

Working Sample

Primary sample [Reserved Key](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_RESERVED_KEY](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_SCENERY_COMPLEXITY

The [SIMCONNECT_RECV_SCENERY_COMPLEXITY](#) structure is used with the [SimConnect_RequestSceneryComplexity](#) function to retrieve the current scenery complexity setting.

Syntax

```
struct SIMCONNECT_RECV_SCENERY_COMPLEXITY : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwSceneryComplexity;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwSceneryComplexity

Double word containing the current scenery complexity. See ENUM [SIMCONNECT_SCENERY_COMPLEXITY](#).

Working Samples

Primary samples [Scenery Complexity and Shadow Flags](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_SCENERY_COMPLEXITY](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_SCENERY_COMPLEXITY](#)
-

SIMCONNECT_RECV_SHADOW_FLAGS

The [SIMCONNECT_RECV_SHADOW_FLAGS](#) structure is used with the [SimConnect_RequestShadowFlags](#) function to retrieve the current shadow settings.

Syntax

```
struct SIMCONNECT_RECV_SHADOW_FLAGS : public SIMCONNECT_RECV {
    DWORD dwRequestID;
    DWORD dwShadowFlags;
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwShadowFlags

Double word containing the current shadow flag settings. See ENUM [SIMCONNECT_SHADOW_FLAGS](#).

Working Samples

Primary samples [Scenery Complexity and Shadow Flags](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_SHADOW_FLAGS](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_SHADOW_FLAGS](#)
-

SIMCONNECT_RECV_SESSION_DURATION

The [SIMCONNECT_RECV_SESSION_DURATION](#) structure is used with the [SimConnect_RequestSessionDuration](#) function to request the simulated time in seconds since the last scenario load. When in a scenario, the duration is accumulated between scenario saves/loads, such as saving and loading checkpoints.

Syntax

```
struct SIMCONNECT_RECV_SESSION_DURATION : public SIMCONNECT_RECV {
    DWORD dwRequestID;
    double dSessionDuration;
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dSessionDuration

A 64-bit floating point number containing the current duration in seconds.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_SESSION_DURATION](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

-
- [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_SYNCHRONOUS_BLOCK

The **SIMCONNECT_RECV_SYNCHRONOUS_BLOCK** structure is sent in response to a call to [SimConnect_RequestSynchronousBlock](#) function.

Syntax

```
struct SIMCONNECT_RECV_SYNCHRONOUS_BLOCK : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwFlags;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwFlags

Double word containing a subset of the **SIMCONNECT_DATA_REQUEST_FLAG** enum (currently only the **SIMCONNECT_DATA_REQUEST_FLAG_BLOCK** is supported).

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_SYNCHRONOUS_BLOCK**.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestSynchronousBlock](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_SYSTEM_STATE

The **SIMCONNECT_RECV_SYSTEM_STATE** structure is used with the [SimConnect_RequestSystemState](#) function to retrieve specific Prepar3D systems states and information.

Syntax

```
struct SIMCONNECT_RECV_SYSTEM_STATE : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwInteger;  
    float fFloat;  
    char szString[MAX_PATH];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwInteger

Double word containing an integer, or boolean, value.

fFloat

A float value.

szString

Null-terminated string.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_SYSTEM_STATE](#).

Typically only one of the received integer, float or string will contain information, which one will depend on the request and can be identified by the request ID. Refer to the descriptions of the [SimConnect_SetSystemState](#) and [SimConnect_RequestSystemState](#) functions.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_SYSTEM_STATE_W

The [SIMCONNECT_RECV_SYSTEM_STATE_W](#) structure is used with the [SimConnect_RequestSystemStateW](#) function to retrieve specific *Prepar3D* systems states and information.

Syntax

```
struct SIMCONNECT_RECV_SYSTEM_STATE_W : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwInteger;  
    float fFloat;  
    wchar_t szString[MAX_PATH];  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwInteger

Double word containing an integer, or boolean, value.

fFloat

A float value.

szString

Null-terminated string.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_SYSTEM_STATE_W](#).

Typically only one of the received integer, float or string will contain information, which one will depend on the request and can be identified by the request ID. Refer to the descriptions of the [SimConnect_SetSystemState](#) and [SimConnect_RequestSystemStateW](#) functions.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_TACAN_LIST

The **SIMCONNECT_RECV_TACAN_LIST** structure is used to return a list of **SIMCONNECT_DATA_FACILITY_TACAN** structures.

Syntax

```
struct SIMCONNECT_RECV_TACAN_LIST : public  
SIMCONNECT_RECV_FACILITIES_LIST{  
    SIMCONNECT_DATA_FACILITY_TACAN rgData[1];  
};
```

Members

rgData[1]
Array of **SIMCONNECT_DATA_FACILITY_TACAN** structures.

Remarks

This structure inherits the **SIMCONNECT_RECV_FACILITIES_LIST** structure, which identifies the number of elements in the list, and the number of packets needed to transmit all the data.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect_RequestFacilitiesList](#)
 - [SIMCONNECT_RECV_AIRPORT_LIST](#)
 - [SIMCONNECT_RECV_FACILITIES_LIST](#)
 - [SIMCONNECT_RECV_NDB_LIST](#)
 - [SIMCONNECT_RECV_WAYPOINT_LIST](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_TRAFFIC_SETTINGS

The **SIMCONNECT_RECV_TRAFFIC_SETTINGS** structure is used with the [SimConnect_RequestTrafficSettings](#) function to retrieve the current traffic settings.

Syntax

```
struct SIMCONNECT_RECV_TRAFFIC_SETTINGS : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    UINT uAirlineDensity;  
    UINT uGADensity;  
    UINT uRoadTrafficDensity;  
    UINT uShipsAndFerriesDensity;  
    UINT uLeisureBoatDensity;  
    SIMCONNECT_DYNAMIC_FREQUENCY eAirportVehicleDensity;  
    BOOL bIFROOnly;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

uAirlineDensity

Unsigned integer containing the current airline traffic density from 0 to 100.

uGADensity

Unsigned integer containing the current general aviation traffic density from 0 to 100.

uRoadTrafficDensity

Unsigned integer containing the current road traffic density from 0 to 100.

uShipsAndFerriesDensity

Unsigned integer containing the current ship and ferry density from 0 to 100.

uLeisureBoatDensity

Unsigned integer containing the current leisure boat traffic density from 0 to 100.

eAirportVehicleDensity

Enumeration containing the current airport vehicle traffic density. See also the enumeration

[**SIMCONNECT_DYNAMIC_FREQUENCY**](#)

bIFROnly

Boolean representing if only IFR traffic should be present.

Working Samples

Primary samples None

Remarks

This structure inherits the [**SIMCONNECT_RECV**](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_TRAFFIC_SETTINGS**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_DYNAMIC_FREQUENCY](#)
-

SIMCONNECT_RECV_VERSION

The **SIMCONNECT_RECV_VERSION** structure is used with the [SimConnect_RequestVersion](#) function to retrieve license type, Prepar3D version, and Prepar3D SimConnect version.

Syntax

```
struct SIMCONNECT_RECV_VERSION : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    DWORD dwApplicationVersionType;  
    DWORD dwApplicationVersionMajor;  
    DWORD dwApplicationVersionMinor;  
    DWORD dwApplicationBuildMajor;  
    DWORD dwApplicationBuildMinor;  
    DWORD dwSimConnectVersionMajor;  
    DWORD dwSimConnectVersionMinor;  
    DWORD dwSimConnectBuildMajor;  
    DWORD dwSimConnectBuildMinor;  
};
```

Members

dwRequestID

Double word containing the client defined request ID.

dwApplicationVersionType

Double word containing Application Version Type. See ENUM [SIMCONNECT_LICENSE_TYPE](#).

dwApplicationVersionMajor

Double word containing Application Version Major.

dwApplicationVersionMinor

Double word containing Application Build Minor.

dwApplicationBuildMajor

Double word containing Application Build Major.

dwApplicationBuildMinor

Double word containing Application Version Minor.

dwSimConnectVersionMajor

Double word containing SimConnect Version Major.

dwSimConnectVersionMinor

Double word containing SimConnect Version Minor.

dwSimConnectBuildMajor

Double word containing SimConnect Build Major.

dwSimConnectBuildMinor

Double word containing SimConnect Build Minor.

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the dwID parameter of [SIMCONNECT_RECV](#) is set to [SIMCONNECT_RECV_ID_VERSION](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
-

SIMCONNECT_RECV_VOR_LIST

The **SIMCONNECT_RECV_VOR_LIST** structure is used to return a list of [SIMCONNECT_DATA_FACILITY_VOR](#) structures.

Syntax

```
struct SIMCONNECT_RECV_VOR_LIST : public SIMCONNECT_RECV_FACILITIES_LIST{  
    SIMCONNECT_DATA_FACILITY_VOR rgData[1];  
};
```

Members**rgData[1]**

Array of [SIMCONNECT_DATA_FACILITY_VOR](#) structures.

Remarks

This structure inherits the [SIMCONNECT_RECV_FACILITIES_LIST](#) structure, which identifies the number of elements in the list, and the number of packets needed to transmit all the data.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect_RequestFacilitiesList](#)
 - [SIMCONNECT_RECV_AIRPORT_LIST](#)
 - [SIMCONNECT_RECV_FACILITIES_LIST](#)
 - [SIMCONNECT_RECV_NDB_LIST](#)
 - [SIMCONNECT_RECV_WAYPOINT_LIST](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_WAYPOINT_LIST

The **SIMCONNECT_RECV_WAYPOINT_LIST** structure is used to return a list of [SIMCONNECT_DATA_FACILITY_WAYPOINT](#) structures.

Syntax

```
struct SIMCONNECT_RECV_WAYPOINT_LIST : public  
SIMCONNECT_RECV_FACILITIES_LIST{  
    SIMCONNECT_DATAFacility_WAYPOINT rgData[1];  
};
```

Members

rgData[1]
Array of [SIMCONNECT_DATA_FACILITY_WAYPOINT](#) structures.

Remarks

This structure inherits the [SIMCONNECT_RECV_FACILITIES_LIST](#) structure, which identifies the number of elements in the list, and the number of packets needed to transmit all the data.

See the remarks for [SimConnect_RequestFacilitiesList](#).

See Also

- [SimConnect_RequestFacilitiesList](#)
 - [SIMCONNECT_RECV_AIRPORT_LIST](#)
 - [SIMCONNECT_RECV_FACILITIES_LIST](#)
 - [SIMCONNECT_RECV_NDB_LIST](#)
 - [SIMCONNECT_RECV_TACAN_LIST](#)
 - [SIMCONNECT_RECV_VOR_LIST](#)
 - [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
-

SIMCONNECT_RECV_WEATHER_OBSERVATION

The **SIMCONNECT_RECV_WEATHER_OBSERVATION** structure is used to return weather observation data, after calls to one of: [SimConnect_WeatherRequestInterpolatedObservation](#), [SimConnect_WeatherRequestObservationAtStation](#), or [SimConnect_WeatherRequestObservationAtNearestStation](#)

Syntax

```
struct SIMCONNECT_RECV_WEATHER_OBSERVATION : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    char szMETAR[1];  
};
```

Members

dwRequestID

The ID of the client defined request.

szMETAR[1]

Null-terminated string containing the Metar weather data. The maximum length of this string is 2000 chars. See the section [Metar Data Format](#) for details on the format required.

Working Sample

Primary sample [Weather Station](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_WEATHER_OBSERVATION**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV_CLOUD_STATE](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_RECV_EVENT](#)
 - [SIMCONNECT_RECV_EXCEPTION](#)
 - [SIMCONNECT_RECV_ID](#)
 - [SIMCONNECT_RECV_SIMOBJECT_DATA](#)
-

SIMCONNECT_RECV_OBSERVER_DATA

The **SIMCONNECT_RECV_OBSERVER_DATA** structure is used to return observer data, after calls to [SimConnect_RequestObserverData](#).

Syntax

```
struct SIMCONNECT_RECV_OBSERVER_DATA : public SIMCONNECT_RECV {  
    DWORD dwRequestID;  
    char szObserverName[128];  
    SIMCONNECT_DATA_OBSERVER ObserverData;  
};
```

Members

dwRequestID

The ID of the client defined request.

szObserverName[128]

The name of the observer.

ObserverData

The all data for the observer.

Working Sample

Primary sample [ManagedObserverControl](#)

Remarks

This structure inherits the [SIMCONNECT_RECV](#) structure and is returned when the **dwID** parameter of **SIMCONNECT_RECV** is set to **SIMCONNECT_RECV_ID_OBSERVER_DATA**.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_RECV](#)
 - [SIMCONNECT_DATA_OBSERVER](#)
-

SIMCONNECT_SIMOBJECT_TYPE

The **SIMCONNECT_SIMOBJECT_TYPE** enumeration type is used with the [SimConnect_RequestDataOnSimObjectType](#) call to request information on specific or nearby objects.

Syntax

```
enum SIMCONNECT_SIMOBJECT_TYPE{  
    SIMCONNECT_SIMOBJECT_TYPE_USER,  
    SIMCONNECT_SIMOBJECT_TYPE_ALL,
```

```
SIMCONNECT_SIMOBJECT_TYPE_AIRPLANE,
SIMCONNECT_SIMOBJECT_TYPE_AIRCRAFT =
SIMCONNECT_SIMOBJECT_TYPE_AIRPLANE,
SIMCONNECT_SIMOBJECT_TYPE_HELICOPTER,
SIMCONNECT_SIMOBJECT_TYPE_BOAT,
SIMCONNECT_SIMOBJECT_TYPE_GROUND
SIMCONNECT_SIMOBJECT_TYPE_WEAPON
SIMCONNECT_SIMOBJECT_TYPE_COUNTERMEASURE
SIMCONNECT_SIMOBJECT_TYPE_ANIMAL
SIMCONNECT_SIMOBJECT_TYPE_AVATAR
SIMCONNECT_SIMOBJECT_TYPE_BLIMP
SIMCONNECT_SIMOBJECT_TYPE_CONTROL_TOWER
SIMCONNECT_SIMOBJECT_TYPE_EXTERNAL_SIM
SIMCONNECT_SIMOBJECT_TYPE_SIMPLE_OBJECT
SIMCONNECT_SIMOBJECT_TYPE_SUBMERSIBLE
SIMCONNECT_SIMOBJECT_TYPE_VIEWER
};
```

Members

SIMCONNECT_SIMOBJECT_TYPE_USER

Specifies the user's aircraft.

SIMCONNECT_SIMOBJECT_TYPE_ALL

Specifies all objects.

SIMCONNECT_SIMOBJECT_TYPE_AIRCRAFT

Specifies all airplanes. Same as **SIMCONNECT_SIMOBJECT_TYPE_AIRPLANE**.

SIMCONNECT_SIMOBJECT_TYPE_AIRPLANE

Specifies all airplanes. Same as **SIMCONNECT_SIMOBJECT_TYPE_AIRCRAFT**.

SIMCONNECT_SIMOBJECT_TYPE_HELICOPTER

Specifies all helicopters.

SIMCONNECT_SIMOBJECT_TYPE_BOAT

Specifies all boats.

SIMCONNECT_SIMOBJECT_TYPE_GROUND

Specifies all ground vehicles.

SIMCONNECT_SIMOBJECT_TYPE_WEAPON

Specifies all weapons.

SIMCONNECT_SIMOBJECT_TYPE_COUNTERMEASURE

Specifies all countermeasures.

SIMCONNECT_SIMOBJECT_TYPE_ANIMAL

Specifies all animals.

SIMCONNECT_SIMOBJECT_TYPE_AVATAR

Specifies all avatars.

SIMCONNECT_SIMOBJECT_TYPE_BLIMP

Specifies all blimps.

SIMCONNECT_SIMOBJECT_TYPE_CONTROL_TOWER

Specifies all control towers.

SIMCONNECT_SIMOBJECT_TYPE_EXTERNAL_SIM

Specifies all ExternalSim objects.

SIMCONNECT_SIMOBJECT_TYPE_SIMPLE_OBJECT

Specifies all simple objects.

SIMCONNECT_SIMOBJECT_TYPE_SUBMERSIBLE

Specifies all submersibles.

SIMCONNECT_SIMOBJECT_TYPE_VIEWER

Specifies all viewer objects.

Working Samples

Primary samples [Request Data](#)

[Set Data](#)

Reference samples [Tagged Data](#)
[Throttle Control](#)

Remarks

This enum is used to specify the return of the object IDs of all objects created using the AI creation functions, whether they are created by this client, other clients, or *Prepar3D* itself. It can also be used to specify the return the object ID of the user aircraft. However it cannot be used to specify the IDs of objects like cars moving on freeways, which are not controlled by the AI component.

See the remarks and examples for [SimConnect_AddToDataDefinition](#).

See Also

- [SimConnect API Reference](#)
 - [SimConnect_RequestDataOnSimObjectType](#)
 - [SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE](#)
-

SIMCONNECT_STATE

The **SIMCONNECT_STATE** enumeration type is used with the [SimConnect_SetSystemEventState](#) call to turn the reporting of events on and off.

Syntax

```
enum SIMCONNECT_STATE{  
    SIMCONNECT_STATE_OFF,  
    SIMCONNECT_STATE_ON  
};
```

Members

SIMCONNECT_STATE_OFF

Specifies off.

SIMCONNECT_STATE_ON

Specifies on.

Working Samples

[Cockpit Camera](#)

Primary samples [Input Event](#)
[Joystick Input](#)
[Throttle Control](#)

Reference samples [Set Data](#)

Remarks

See the remarks for [SimConnect_MapInputEventToClientEvent](#).

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetSystemEventState](#)
-

SIMCONNECT_DATA_INITPOSITION

The **SIMCONNECT_DATA_INITPOSITION** structure is used to initialize the position of the user aircraft, AI controlled aircraft, or other simulation object.

Syntax

```
struct SIMCONNECT_DATA_INITPOSITION{
    double Latitude;
    double Longitude;
    double Altitude;
    double Pitch;
    double Bank;
    double Heading;
    DWORD OnGround;
    DWORD Airspeed;
};
```

Members

Latitude

Latitude in degrees.

Longitude

Longitude in degrees.

Altitude

Altitude in feet.

Pitch

Pitch in degrees.

Bank

Bank in degrees.

Heading

Heading in degrees.

OnGround

Set this to 1 to place the object on the ground, or 0 if the object is to be airborne.

Airspeed

The airspeed in knots, or one of the following special values:

Define	Value	Description
INITPOSITION_AIRSPEED_CRUISE	-1	The aircraft's design cruising speed.
INITPOSITION_AIRSPEED_KEEP	-2	Maintain the current airspeed.

Working Samples

[AI Objects and Waypoints](#)

[Set Data](#)

Primary samples

[Managed AI Waypoints](#)

Remarks

The primary use of this structure is to *initialize* the positioning of the user aircraft, because it also

optimizes some of the terrain and other simulation systems. Simply setting parameters such as latitude, longitude and altitude does not perform this kind of optimization. This structure should not be used to incrementally move the user aircraft (as this will unnecessarily initiate the reloading of scenery), in this case change the latitude, longitude, altitude and other parameters of the aircraft appropriately (using the variables described in the [Simulation Variables](#) document).

This structure can be used to incrementally move or reposition an AI controlled aircraft, or any other aircraft not controlled by the user, as the terrain system optimizations are not performed in this case.

This structure is used by the functions: [SimConnect_AICreateNonATCAircraft](#), [SimConnect_AICreateSimulatedObject](#) and [SimConnect_AddToDataDefinition](#).

This structure can only be used to set data, it cannot be used as part of a data request.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_DATATYPE](#)
-

SIMCONNECT_DATA_LATLONALT

The **SIMCONNECT_DATA_LATLONALT** structure is used to hold a world position.

Syntax

```
struct SIMCONNECT_DATA_LATLONALT{
    double Latitude;
    double Longitude;
    double Altitude;
};
```

Members

Latitude

The latitude of the position in degrees.

Longitude

The longitude of the position in degrees.

Altitude

The altitude of the position in feet.

Remarks

This structure is used when one of the following simulation variables is requested (with a call to [SimConnect_RequestDataOnSimObject](#)):

- ADF LATLONALT
- NAV DME LATLONALT
- NAV GS LATLONALT
- NAV VOR LATLONALT
- TACAN LATLONALT
- INNER MARKER LATLONALT
- MIDDLE MARKER LATLONALT
- OUTER MARKER LATLONALT

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_DATATYPE](#)
-

SIMCONNECT_DATA_MARKERSTATE

The **SIMCONNECT_DATA_MARKERSTATE** structure is used to help graphically link flight model data with the graphics model.

Syntax

```
struct SIMCONNECT_DATA_MARKERSTATE{
    char szMarkerName[64];
    DWORD dwMarkerState;
};
```

Members

szMarkerName[64]

Null-terminated string containing the marker name. One from the following table:

String

Cg
ModelCenter
Wheel
Skid
Ski
Float
Scrape
Engine
Prop
Eyepoint
LongScale
LatScale
VertScale
AeroCenter
WingApex
RefChord
Datum
WingTip
FuelTank
Forces

dwMarkerState

Double word containing the marker state, set to 1 for on and 0 for off.

Remarks

The [SimConnect_AddToDataDefinition](#) call can be used to add a **SIMCONNECT_DATA_MARKERSTATE** structure to a data definition. Use of this call and structure is to help determine that points specified in the flight model of an aircraft match the graphics model for that aircraft, by turning on the specified marker lights. A SimConnect client created to do this becomes a tool to aid to the accurate development of aircraft models, rather than an add-on that an end user might run.

This structure can only be used as input, it cannot be used as part of a data request.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_DATATYPE](#)
 - [SIMCONNECT_DATA_INITPOSITION](#)
-

SIMCONNECT_DATA_WAYPOINT

The **SIMCONNECT_DATA_WAYPOINT** structure is used to hold all the necessary information on a waypoint.

Syntax

```
struct SIMCONNECT_DATA_WAYPOINT{
    double Latitude;
    double Longitude;
    double Altitude;
    unsigned long Flags;
    double ktsSpeed;
    double percentThrottle;
};
```

Members

Latitude

The latitude of the waypoint in degrees.

Longitude

The longitude of the waypoint in degrees.

Altitude

The altitude of the waypoint in feet.

Flags

Specifies the flags set for this waypoint, see [SIMCONNECT_WAYPOINT_FLAGS](#). These flags can be OR'ed together, for example:

Flags = SIMCONNECT_WAYPOINT_ON_GROUND | SIMCONNECT_WAYPOINT_REVERSE;

ktsSpeed

Specifies the required speed in knots. If a specific speed is required, then the **SIMCONNECT_WAYPOINT_SPEED_REQUESTED** flag must be set to **True**.

percentThrottle

Specifies the required throttle as a percentage. If a specific throttle percentage is required, then the **SIMCONNECT_THROTTLE_REQUESTED** flag must be set to **True**.

Working Sample

Primary sample [AI Objects and Waypoints](#)
[Managed AI Waypoints](#)

Remarks

The [SimConnect_AddToDataDefinition](#) call can be used to add a **SIMCONNECT_DATA_WAYPOINT** structure to a data definition. A list of waypoints is sent to an AI object using the [SimConnect_SetDataOnSimObject](#) function. There is no limit to the number of waypoints that can be sent to an object. If just one waypoint is set, the **SIMCONNECT_WAYPOINT_WRAP_TO_FIRST** flag should not be used.

If a speed is requested at a waypoint, the slower that speed is the closer the object will approach the exact point of the waypoint, requests for high speeds can result in the AI system turning the object some way off of the waypoint. The pitch, bank and heading of objects controlled by the waypoint system are determined by the AI pilot, and cannot be set from a client.

Because of the curvature of the Earth it is recommended that waypoints for AI controlled objects are not more than five miles apart, as if the object is flying it will fly *straight* from one waypoint to the next -- and hence loose altitude if the waypoints are some distance apart.

This structure can only be used to set data, it cannot be used as part of a data request.

See Also

-
- [SimConnect API Reference](#)
 - [SIMCONNECT_DATATYPE](#)
-

SIMCONNECT_DATA_XYZ

The **SIMCONNECT_DATA_XYZ** structure is used to hold a 3D co-ordinate.

Syntax

```
struct SIMCONNECT_DATA_XYZ{  
    double x;  
    double y;  
    double z;  
};
```

Members

- x**
The position along the x axis.
- y**
The position along the y axis.
- z**
The position along the z axis.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_DATATYPE](#)
-

SIMCONNECT_DATA_PBH

The **SIMCONNECT_DATA_PBH** structure is used to hold the pitch, bank, and heading.

Syntax

```
struct SIMCONNECT_DATA_PBH{  
    double Pitch;  
    double Bank;  
    double Heading;  
};
```

Members

- Pitch**
The rotation for the pitch.
- Bank**
The rotation for the bank.
- Heading**
The rotation for the heading.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_DATATYPE](#)
-

SIMCONNECT_DATA_OBSERVER

The **SIMCONNECT_DATA_OBSERVER** structure is used to hold all the necessary information on

an observer.

Syntax

```
struct SIMCONNECT_DATA_OBSERVER{
    SIMCONNECT_DATA_LATLONALT Position;
    SIMCONNECT_DATA_PBH Rotation;
    DWORD Regime;
    BOOL RotateOnTarget;
    BOOL FocusFixed;
    float FocalLength;
    float FieldOfViewH;
    float FieldOfViewV;
    float LinearStep;
    float AngularStep;
};
```

Members

Position

The world location (latitude/longitude/altitude) of the observer.

Rotation

The rotation (pitch/bank/heading) of the observer in degrees.

Regime

The restrictions of the observer. Tellurian (earth-based) is 0, Terrestrial (land-based) is 1, and Ghost (unimpeded) is 2.

RotateOnTarget

Specifies if the observer should rotate on target. Observer will rotate about its own origin if **false** and rotate about its focal point (target) if **true**.

FocusFixed

Specifies the observer focal point. If **false**, the observer automatically shifts focal point to the world (terrain) as the observer is manipulated. If **true**, the observer locks the focus to a fixed distance relative to the observer's position.

FocalLength

Specifies the focal length in meters.

FieldOfViewH

Specifies the horizontal field of view in degrees.

FieldOfViewV

Specifies the vertical field of view in degrees.

LinearStep

Specifies the linear step in meters.

AngularStep

Specifies the angular step in degrees.

Working Sample

Primary sample [Managed Observer Control](#)

Remarks

None.

See Also

- [SimConnect_CreateObserver](#)
- [SimConnect_RequestObserverData](#)

SIMCONNECT_DATA_OBJECT_DAMAGED_BY_WEAPON

The **SIMCONNECT_DATA_OBJECT_DAMAGED_BY_WEAPON** structure is used to hold data

when an object is damaged by a weapon.

Syntax

```
struct SIMCONNECT_DATA_OBJECT_DAMAGED_BY_WEAPON {
    DWORD dwWeaponID;
    DWORD dwAttackerID;
    DWORD dwDamagedObjectID;
};
```

Members

dwWeaponID

Double word containing the object ID of the weapon.

dwAttackerID

Double word containing the object ID of the object that fired the weapon.

dwDamagedObjectID

Double word containing the object ID of the damaged entity.

Remarks

None.

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_DATATYPE](#)
 - [SIMCONNECT_RECV_EVENT_WEAPON](#)
 - [SIMCONNECT_RECV_EVENT_COUNTERMEASURE](#)
 - [SIMCONNECT_RECV_EVENT_OBJECT_DAMAGED_BY_WEAPON](#)
-

SIMCONNECT_DATA_VIDEO_STREAM_INFO

The **SIMCONNECT_DATA_VIDEO_STREAM_INFO** structure is used to hold data for a video stream.

Syntax

```
struct SIMCONNECT_DATA_VIDEO_STREAM_INFO {
    char szSourceAddress[48];
    char szDestinationAddress[48];
    UINT uPort;
    UINT uWidth;
    UINT uHeight;
    UINT uFramerate;
    UINT uBitRate;
    UINT uFormat;
};
```

Members

szSourceAddress

IP Address of the video stream source. IPv4 address in standard dotted-decimal notation.

szDestinationAddress

IP Address of the video stream destination. IPv4 address in standard dotted-decimal notation.

uPort

Port of the video stream destination. This must be an even number. Source port will be (uPort -1).

uWidth

Width of the video in pixels.

uHeight

Height of the video in pixels.

uFramerate

Target framerate to use for video encoding. Standard video framerates such as 20, 24, or 30 are recommended. are recommended.

uBitRate

Target bitrate (bps) to use for video encoding. Recommended range is 500,000-20,000,000.

uFormat

Format to use for video encoding. Formats: 0 - H.264.

Remarks

Every rendered frame in Prepar3D will be encoded for streaming. To keep frame quality stable, the frame length used for each encoded frame will be ((1 second)/uFramerate). For this reason uFramerate and uBitRate directly effect the quality of each frame, but may not reflect the final streaming bitrate if Prepar3D's framerate differs from the target framerate used for encoding.

Here are some bitrate ranges used by commercial web streaming services:

480p/30fps: 1,000,000-1,7000,000 bps

720p/30fps: 2,000,000-3,6000,000 bps

1080p/30fps: 3,500,000-5,000,000 bps.

uFormat is currently ignored because only one format is supported. More formats may be added in the future, so this value should be set to 0.

If the size of the Prepar3D window being streamed does not match uWidth and uHeight, the rendered image will be upscaled or downscaled to the video stream resolution.

There are several streaming settings in the graphics section of the Prepar3D.cfg:

- **VIDEO_CAPTURE_WIDTH** (default is 480): Default width value used if uWidth provided is 0
- **VIDEO_CAPTURE_HEIGHT** (default is 360): Default width value used if uHeight provided is 0
- **VIDEO_CAPTURE_FPS** (default is 30): Default framerate value used if uFramerate provided is 0
- **VIDEO_CAPTURE_BIT_RATE** (default is 800000): Default bitrate value used if uBitRate provided is 0
- **VIDEO_CAPTURE_TTL** (default is 255): Time To Live value used for all stream connections
- **VIDEO_CAPTURE_IS_THREADED** (default is True): Perform video encoding on a background thread.
- **VIDEO_CAPTURE_INFO_SEND_RATE** (default is 20): Number of frames between repeated sending of stream info. This allows clients to join midstream.
- **VIDEO_CAPTURE_MAX_FRAME_SIZE** (default is 5000000): Max buffer size in bytes for a streamed frame

Here is an example session description protocol (sdp) file for viewing a Prepar3D stream sent to localhost:

v=0
o=- 0 IN IP4 127.0.0.1
s=No Name
t=0 0
m=video 1234 RTP/AVP 96
c=IN IP4 127.0.0.1
a=rtpmap:96 H264/90000
a=fmt:96 packetization - mode = 1

See Also

- [SimConnect_BeginVideoStream](#)
- [SimConnect_EndVideoStream](#)
- [SimConnect API Reference](#)

-
- [SIMCONNECT_DATATYPE](#)
-

SIMCONNECT_TEXT_RESULT

The **SIMCONNECT_TEXT_RESULT** enumeration type is used to specify which event has occurred as a result of a call to [SimConnect_Text](#).

Syntax

```
enum SIMCONNECT_TEXT_RESULT{
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_1,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_2,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_3,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_4,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_5,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_6,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_7,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_8,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_9,
    SIMCONNECT_TEXT_RESULT_MENU_SELECT_10,
    SIMCONNECT_TEXT_RESULT_DISPLAYED =0x00010000,
    SIMCONNECT_TEXT_RESULT_QUEUED,
    SIMCONNECT_TEXT_RESULT_REMOVED,
    SIMCONNECT_TEXT_RESULT_REPLACE,
    SIMCONNECT_TEXT_RESULT_TIMEOUT
};
```

Members

SIMCONNECT_TEXT_RESULT_MENU_SELECT_1 to SIMCONNECT_TEXT_RESULT_MENU_SELECT_10

Specifies that the user has selected the menu item.

SIMCONNECT_TEXT_RESULT_DISPLAYED

Specifies that the menu or text identified by the EventID is now on display.

SIMCONNECT_TEXT_RESULT_QUEUED

Specifies that the menu or text identified by the EventID is waiting in a queue.

SIMCONNECT_TEXT_RESULT_REMOVED

Specifies that the menu or text identified by the EventID has been removed from the queue.

SIMCONNECT_TEXT_RESULT_REPLACE

Specifies that the menu or text identified by the EventID has been replaced in the queue.

SIMCONNECT_TEXT_RESULT_TIMEOUT

Specifies that the menu or text identified by the EventID has timed-out and is no longer on display.

Working Sample

Primary sample [Text Menu](#)

Remarks

See the remarks for [SimConnect_Text](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_TEXT_TYPE](#)
-

SIMCONNECT_TEXT_TYPE

The **SIMCONNECT_TEXT_TYPE** enumeration type is used to specify which type of text is to be displayed by the [SimConnect_Text](#) function

Syntax

```
enum SIMCONNECT_TEXT_TYPE{
    SIMCONNECT_TEXT_TYPE_SCROLL_BLACK,
    SIMCONNECT_TEXT_TYPE_SCROLL_WHITE,
    SIMCONNECT_TEXT_TYPE_SCROLL_RED,
    SIMCONNECT_TEXT_TYPE_SCROLL_GREEN,
    SIMCONNECT_TEXT_TYPE_SCROLL_BLUE,
    SIMCONNECT_TEXT_TYPE_SCROLL_YELLOW,
    SIMCONNECT_TEXT_TYPE_SCROLL_MAGENTA,
    SIMCONNECT_TEXT_TYPE_SCROLL_CYAN,
    SIMCONNECT_TEXT_TYPE_PRINT_BLACK = 0x0100,
    SIMCONNECT_TEXT_TYPE_PRINT_WHITE,
    SIMCONNECT_TEXT_TYPE_PRINT_RED,
    SIMCONNECT_TEXT_TYPE_PRINT_GREEN,
    SIMCONNECT_TEXT_TYPE_PRINT_BLUE,
    SIMCONNECT_TEXT_TYPE_PRINT_YELLOW,
    SIMCONNECT_TEXT_TYPE_PRINT_MAGENTA,
    SIMCONNECT_TEXT_TYPE_PRINT_CYAN,
    SIMCONNECT_TEXT_TYPE_MENU = 0x0200
    SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW = 0x0300
};
```

Members

SIMCONNECT_TEXT_TYPE_SCROLL_BLACK to SIMCONNECT_TEXT_TYPE_SCROLL_CYAN

Specifies scrolling text in the named color.

SIMCONNECT_TEXT_TYPE_PRINT_BLACK to SIMCONNECT_TEXT_TYPE_PRINT_CYAN

Specifies static text in the named color.

SIMCONNECT_TEXT_TYPE_MENU

Specifies that the text is for a menu.

SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW

Specifies that the text is for a message window.

Working Sample

Primary sample [Text Menu](#)

Remarks

See the remarks for [SimConnect_Text](#).

See Also

- [SimConnect API Reference](#)
 - [SIMCONNECT_TEXT_RESULT](#)
-

SIMCONNECT_WAYPOINT_FLAGS

The **SIMCONNECT_WAYPOINT_FLAGS** enumeration type is used with the [SIMCONNECT_DATA_WAYPOINT](#) structure to define waypoints.

Syntax

```
enum SIMCONNECT_WAYPOINT_FLAGS{
    SIMCONNECT_WAYPOINT_SPEED_REQUESTED = 0x04,
    SIMCONNECT_WAYPOINT_THROTTLE_REQUESTED = 0x08,
    SIMCONNECT_WAYPOINT_COMPUTE_VERTICAL_SPEED = 0x10,
    SIMCONNECT_WAYPOINT_ALTITUDE_IS_AGL = 0x20,
    SIMCONNECT_WAYPOINT_ON_GROUND = 0x00100000,
    SIMCONNECT_WAYPOINT_REVERSE = 0x00200000,
    SIMCONNECT_WAYPOINT_WRAP_TO_FIRST = 0x00400000,
};
```

Members

SIMCONNECT_WAYPOINT_SPEED_REQUESTED

Specifies requested speed is valid.

SIMCONNECT_WAYPOINT_THROTTLE_REQUESTED

Specifies requested throttle percentage is valid.

SIMCONNECT_WAYPOINT_COMPUTE_VERTICAL_SPEED

Specifies that the vertical should be calculated to reach the required speed when crossing the waypoint.

SIMCONNECT_WAYPOINT_ALTITUDE_IS_AGL

Specifies the altitude specified is AGL (above ground level).

SIMCONNECT_WAYPOINT_ON_GROUND

Specifies the waypoint should be on the ground. Make sure this flag is set if the aircraft is to taxi to this point.

SIMCONNECT_WAYPOINT_REVERSE

Specifies that the aircraft should back up to this waypoint. This is only valid on the first waypoint.

SIMCONNECT WAYPOINT WRAP TO FIRST

Specifies that the next waypoint is the first waypoint. This is only valid on the last waypoint.

Working Sample

Primary sample [AI Objects and Waypoints](#)
[Managed AI Waypoints](#)

Remarks

To set multiple waypoint flags simply OR them together. See the remarks for the [SIMCONNECT_DATA_WAYPOINT](#) structure.

See Also

- [SimConnect API Reference](#)
-

SIMCONNECT_WEATHER_MODE

The **SIMCONNECT_WEATHER_MODE** enumeration type is used to return the current weather mode, after a call using the **SIMCONNECT_RECV_ID_EVENT_WEATHER_MODE** setting

Syntax

```
enum SIMCONNECT_WEATHER_MODE{
    SIMCONNECT_WEATHER_MODE_THEME,
    SIMCONNECT_WEATHER_MODE_RWW, // deprecated
    SIMCONNECT_WEATHER_MODE_CUSTOM,
    SIMCONNECT_WEATHER_MODE_GLOBAL
};
```

Members

SIMCONNECT_WEATHER_MODE_THEME

Specifies that the weather has been set to a theme.

SIMCONNECT_WEATHER_MODE_RWW

Specifies that real-world weather has been set. This has been deprecated.

SIMCONNECT_WEATHER_MODE_CUSTOM

Specifies that custom weather has been set.

SIMCONNECT_WEATHER_MODE_GLOBAL

Specifies that the global weather mode has been set.

See Also

- [SimConnect API Reference](#)
-

SIMCONNECT_CAMERA_SENSOR_MODE

The **SIMCONNECT_CAMERA_SENSOR_MODE** enumeration type is used with the [SimConnect_SetCameraSensorMode](#) and [SimConnect_SetMainCameraSensorMode](#) call to specify a camera's sensor mode.

Syntax

```
enum SIMCONNECT_CAMERA_SENSOR_MODE{
    SIMCONNECT_CAMERA_SENSOR_NONE,
    SIMCONNECT_CAMERA_SENSOR_IR,
    SIMCONNECT_CAMERA_SENSOR_IR_BLACK_HOT,
    SIMCONNECT_CAMERA_SENSOR_GDATA,
};
```

Members

SIMCONNECT_CAMERA_SENSOR_NONE

No sensor mode.

SIMCONNECT_CAMERA_SENSOR_IR

IR sensor mode.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_SetCameraSensorMode](#)
-

SIMCONNECT_CAMERA_TYPE

The **SIMCONNECT_CAMERA_TYPE** enumeration type is used with the [SimConnect_CreateCameraDefinition](#) call to create a new camera definition.

Syntax

```
enum SIMCONNECT_CAMERA_TYPE{
    SIMCONNECT_CAMERA_TYPE_OBJECT_PANEL,
    SIMCONNECT_CAMERA_TYPE_OBJECT_VIRTUAL,
    SIMCONNECT_CAMERA_TYPE_OBJECT_CENTER,
    SIMCONNECT_CAMERA_TYPE_OBJECT_PILOT,
    SIMCONNECT_CAMERA_TYPE_TOWER,
    SIMCONNECT_CAMERA_TYPE_LATLONALT_ORTHOGONAL,
    SIMCONNECT_CAMERA_TYPE_OBJECT_AI_VIRTUAL
    SIMCONNECT_CAMERA_TYPE_OBJECT_AI_CENTER
};
```

Members

SIMCONNECT_CAMERA_TYPE_OBJECT_PANEL

Specifies the user's cockpit as the camera's origin.

SIMCONNECT_CAMERA_TYPE_OBJECT_VIRTUAL

Specifies the user's virtual cockpit as the camera's origin.

SIMCONNECT_CAMERA_TYPE_OBJECT_CENTER

Specifies the center of the user's vehicle as the camera's origin.

SIMCONNECT_CAMERA_TYPE_OBJECT_PILOT

Specifies the user's pilot as the camera's origin.

SIMCONNECT_CAMERA_TYPE_TOWER

Specifies a control tower as the camera's origin.

SIMCONNECT_CAMERA_TYPE_LATLONALT_ORTHOGONAL

Specifies the center of the user's object as the camera's origin. Viewed from a orthogonal/top-down position.

SIMCONNECT_CAMERA_TYPE_OBJECT_AI_VIRTUAL

Specifies AI object's virtual cockpit as the camera's origin.

SIMCONNECT_CAMERA_TYPE_OBJECT_AI_CENTER

Specifies the center of the AI object's vehicle as the camera's origin.

See Also

- [SimConnect API Reference](#)
 - [SimConnect_CreateCameraDefinition](#)
-

SIMCONNECT_CAMERA_COMMAND

The **SIMCONNECT_CAMERA_COMMAND** enumeration type is used with the [SimConnect_SendCameraCommand](#) call to create move a camera.

Syntax

```
enum SIMCONNECT_CAMERA_COMMAND{
    SIMCONNECT_CAMERA_MOVE_LEFT,
    SIMCONNECT_CAMERA_MOVE_RIGHT,
    SIMCONNECT_CAMERA_MOVE_UP,
    SIMCONNECT_CAMERA_MOVE_DOWN,
    SIMCONNECT_CAMERA_MOVE_FORWARD,
    SIMCONNECT_CAMERA_MOVE_BACK
    SIMCONNECT_CAMERA_PITCH_UP
    SIMCONNECT_CAMERA_PITCH_DOWN
    SIMCONNECT_CAMERA_YAW_LEFT
    SIMCONNECT_CAMERA_YAW_RIGHT
    SIMCONNECT_CAMERA_ROLL_LEFT
    SIMCONNECT_CAMERA_ROLL_RIGHT
    SIMCONNECT_CAMERA_ZOOM_IN
    SIMCONNECT_CAMERA_ZOOM_OUT
    SIMCONNECT_CAMERA_RESET_ROTATION
};
```

Members

SIMCONNECT_CAMERA_MOVE_LEFT

Specifies the user's cockpit as the camera's origin.

SIMCONNECT_CAMERA_MOVE_RIGHT

Specifies the user's virtual cockpit as the camera's origin.

SIMCONNECT_CAMERA_MOVE_UP

Simulate a user input to move the camera up.

SIMCONNECT_CAMERA_MOVE_DOWN

Simulate a user input to move the camera down.

SIMCONNECT_CAMERA_MOVE_FORWARD

Simulate a user input to move the camera forward.

SIMCONNECT_CAMERA_MOVE_BACK

Simulate a user input to move the camera back.

SIMCONNECT_CAMERA_PITCH_UP

Simulate a user input to pitch the camera up.

SIMCONNECT_CAMERA_PITCH_DOWN

Simulate a user input to pitch the camera down.

SIMCONNECT_CAMERA_YAW_LEFT

Simulate a user input to rotate the camera left about the vertical axis.

SIMCONNECT_CAMERA_YAW_RIGHT

Simulate a user input to rotate the camera right about the vertical axis.

SIMCONNECT_CAMERA_ROLL_LEFT

Simulate a user input to rotate the camera left about the longitudinal axis.

SIMCONNECT_CAMERA_ROLL_RIGHT

Simulate a user input to rotate the camera right about the longitudinal axis.

SIMCONNECT_CAMERA_ZOOM_IN

Simulate a user input to zoom the camera in.

SIMCONNECT_CAMERA_ZOOM_OUT

Simulate a user input to zoom the camera out.

SIMCONNECT_CAMERA_RESET_ROTATION

Simulate a user input to rest the camera rotation.

See Also

-
- [SimConnect API Reference](#)
 - [SimConnect_SendCameraCommand](#)

[- top -](#)

PREPAR3D

Key Strings

The following table lists all the strings which can be used to identify keys on the keyboard, for use with the **SimConnect_ReservedKeySet** function, described in the [SimConnect](#) documentation.

Key Strings
"VK_0x00"
"VK_LBUTTON"
"VK_RBUTTON"
"Scroll_Lock"
"VK_MBUTTON"
"VK_XBUTTON1"
"VK_XBUTTON2"
"VK_0x07"
"Backspace"
"Tab"
"VK_0x0A"
"VK_0x0B"
"Num_5"
"Enter"
"VK_0x0E"
"VK_0x0F"
"Shift"
"Ctrl"
"Alt"
"VK_PAUSE"
"Caps_Lock"
"VK_KANA"
"VK_0x16"
"VK_JUNJA"
"VK_FINAL"
"VK_KANJI"
"VK_0x1A"
"Esc"
"VK_CONVERT"
"VK_NONCONVERT"
"VK_ACCEPT"
"VK_MODECHANGE"
"Space"
"Num_9"
"Num_3"
"Num_1"
"Num_7"
"Num_4"
"Num_8"
"Num_6"
"Num_2"
"VK_SELECT"
"VK_PRINT"
"VK_EXECUTE"
"Sys_Req"
"Num_0"
"Num_Del"

"VK_HELP"
"0"
"1"
"2"
"3"
"4"
"5"
"6"
"7"
"8"
"9"
"VK_0x3A"
"VK_0x3B"
"VK_0x3C"
"VK_0x3D"
"VK_0x3E"
"VK_0x3F"
"VK_0x40"
"A"
"B"
"C"
"D"
"E"
"F"
"G"
"H"
"I"
"J"
"K"
"L"
"M"
"N"
"O"
"P"
"Q"
"R"
"S"
"T"
"U"
"V"
"W"
"X"
"Y"
"Z"
"VK_LWIN"
"VK_RWIN"
"VK_APPS"
"VK_0x5E"
"VK_SLEEP"
"VK_NUMPAD0"
"VK_NUMPAD1"
"VK_NUMPAD2"
"VK_NUMPAD3"
"VK_NUMPAD4"
"VK_NUMPAD5"
"VK_NUMPAD6"
"VK_NUMPAD7"
"VK_NUMPAD8"
"VK_NUMPAD9"

"VK_MULTIPLY"
"VK_ADD"
"VK_SEPARATOR"
"VK_SUBTRACT"
"VK_DECIMAL"
"VK_DIVIDE"
"F1"
"F2"
"F3"
"F4"
"F5"
"F6"
"F7"
"F8"
"F9"
"F10"
"F11"
"F12"
"F13"
"F14"
"F15"
"F16"
"F17"
"F18"
"F19"
"F20"
"F21"
"F22"
"F23"
"F24"
"VK_0x88"
"VK_0x89"
"VK_0x8A"
"VK_0x8B"
"VK_0x8C"
"VK_0x8D"
"VK_0x8E"
"VK_0x8F"
"Pause"
"VK_SCROLL"
"VK_OEM_FJ_JISHO"
"VK_OEM_FJ_MASSHOU"
"VK_OEM_FJ_TOUROKU"
"VK_OEM_FJ_LOYA"
"VK_OEM_FJ_ROYA"
"VK_0x97"
"VK_0x98"
"VK_0x99"
"VK_0x9A"
"VK_0x9B"
"VK_0x9C"
"VK_0x9D"
"VK_0x9E"
"VK_0x9F"
"VK_LSHIFT"
"VK_RSHIFT"
"VK_LCONTROL"
"VK_RCONTROL"
"VK_LMENU"

"VK_RMENU"
"VK_BROWSER_BACK"
"VK_BROWSER_FORWARD"
"VK_BROWSER_REFRESH"
"VK_BROWSER_STOP"
"VK_BROWSER_SEARCH"
"VK_BROWSER_FAVORITES"
"VK_BROWSER_HOME"
"VK_VOLUME_MUTE"
"VK_VOLUME_DOWN"
"VK_VOLUME_UP"
"VK_MEDIA_NEXT_TRACK"
"VK_MEDIA_PREV_TRACK"
"VK_MEDIA_STOP"
"VK_MEDIA_PLAY_PAUSE"
"VK_LAUNCH_MAIL"
"VK_LAUNCH_MEDIA_SELECT"
"VK_LAUNCH_APP1"
"VK_LAUNCH_APP2"
"VK_0xB8"
"VK_0xB9"
"VK_SEMICOLON"
"VK_PLUS"
"VK_COMMA"
"VK_MINUS"
"VK_PERIOD"
"VK_SLASH"
"VK_TILDE"
"VK_0xC1"
"VK_0xC2"
"VK_0xC3"
"VK_0xC4"
"VK_0xC5"
"VK_0xC6"
"VK_0xC7"
"VK_0xC8"
"VK_0xC9"
"VK_0xCA"
"VK_0xCB"
"VK_0xCC"
"VK_0xCD"
"VK_0xCE"
"VK_0xCF"
"VK_0xD0"
"VK_0xD1"
"VK_0xD2"
"VK_0xD3"
"VK_0xD4"
"VK_0xD5"
"VK_0xD6"
"VK_0xD7"
"VK_0xD8"
"VK_0xD9"
"VK_0xDA"
"VK_LBRACKET"
"VK_BACKSLASH"
"VK_RBRACKET"
"VK_QUOTE"
"VK_0xDF"

"VK_0xE0"
"VK_OEM_AX"
"VK_OEM_102"
"VK_ICO_HELP"
"VK_ICO_00"
"VK_PROCESSKEY"
"VK_ICO_CLEAR"
"VK_PACKET"
"VK_0xE8"
"VK_OEM_RESET"
"VK_OEM_JUMP"
"VK_OEM_PA1"
"VK_OEM_PA2"
"VK_OEM_PA3"
"VK_OEM_WSCTRL"
"VK_OEM_CUSEL"
"VK_OEM_ATTN"
"VK_OEM_FINISH"
"VK_OEM_COPY"
"VK_OEM_AUTO"
"VK_OEM_ENLW"
"VK_OEM_BACKTAB"
"VK_ATTN"
"VK_CRSEL"
"VK_EXSEL"
"VK_EREOF"
"VK_PLAY"
"VK_ZOOM"
"VK_NONAME"
"VK_PA1"
"VK_OEM_CLEAR"
"VK_0xFF"

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

SimConnect Configuration

Overview

SimConnect has multiple configuration files which allow the developer to configure different aspects of SimConnect. The different configuration files and their purposes are as follows:

- [SimConnect Debug](#): SimConnect.ini
- [SimConnect Client Configuration](#): SimConnect.cfg
- [SimConnect Server Configuration](#): SimConnect.xml

SimConnect Debug (SimConnect.ini)

The default [SimConnect.ini](#) file will enable the debug window and disable the log file. Use a semi-colon to start a comment, or comment out a directive. The file should be placed in your `%USERPROFILE%\Documents\Prepar3D v5 Files` folder.

	Directive	Values	Description
	level	Verbose, Normal, Warning, Error, Off	Set the level of text communication to be provided to the console, debug string, or log file.
	console	Yes, No	Open a command line debug window to display server to client communication.
	RedirectStdOutToConsole	Yes, No	This setting redirects the output of console print functions to the SimConnect console window. This is useful for debugging DLL or GUI based SimConnect applications that do not launch a console window.
	OutputDebugString	Yes, No	Sends server client communication to the debugger for display. Either your SimConnect application or <i>Prepar3D</i> must be launched from within an integrated development environment (IDE) debugger in order for this option to work. Refer to MSDN documentation for more details.
	file	Filename	Output communications to a log file. If the text %03u is included in the filename, then the filename will be incremented each time <i>Prepar3D</i> starts, so giving a new log file for each test run. Example: file=c:\simconnect%03u.log
	file_next_index	Integer	The index of the first log file. Subsequent log files will have the index number incremented by one.

SimConnect Client Configuration (SimConnect.cfg)

The [SimConnect.cfg](#) file contains communications information for a client (the [SimConnect.xml file](#) contains information for a server). This file is only required if a client is going to access *Prepar3D* running on a remote machine, and should be placed in the *Documents* folder, or in the same folder as the client application or library, on the computer the client is running on.

Note: SimConnect.msi is no longer needed due to the fact that the WinSxS library has been deprecated.

The SimConnect.cfg file can contain a number of configurations, identified in sections with the [SimConnect.N] title. The index number is used as a parameter in the [SimConnect_Open](#) function. This is useful for applications that communicate with a number of different machines that are running Prepar3D. The default configuration index is zero, and if there is only one configuration in the file, no index number is required.

Directive	Values	Description
Protocol	Pipe, IPv4, IPv6	Pipe uses a Named-Pipes communication system. The IPv4 and IPv6 protocols are available on computers running Windows, with IPv4 as the default. IPv6 has more security features and is recommended. If IPv6 is not already installed, there is a utility to install it in the config folder.
Address		The order in which protocols are evaluated is: Pipe, IPv6, IPv4, Pipe. So, for example, if this entry is set at IPv6, then IPv4 and Pipe will evaluated in that order. The first working protocol found will be used. The advantage of Pipe communication is that it avoids conflict with firewalls and virus protection software when the connection is local, which of course is not the situation if a SimConnect.cfg file is required.
Port		The Address and Port of the SimConnect server should be entered in these fields, these will be the same values as those in the appropriate Comm section of the Simconnect.xml file . A good knowledge of Windows networking and client/server applications will be needed to set these correctly. The address can be the name of a computer in a Domain Controller environment.
MaxReceiveSize	Integer	The maximum packet size. The default is 8192. If the client receives a packet larger than this size, it will disconnect from the server.
DisableNagle	0, 1	Set to 1 to disable the Nagle packet optimization algorithms.

SimConnect Server Configuration (SimConnect.xml)

The [SimConnect.xml](#) file contains communication information for the SimConnect server (the [SimConnect.cfg file](#) contains information for a client). The default behavior is that three servers are initiated by the system, for local communication using one of each of the three protocols: Pipe, IPv4, and IPv6. These servers do not require entries in the XML file. The Simconnect.xml file is not needed if this covers the communication requirements. If remote connections are required, then one entry will need to be made in this file to cover each type of remote communication that needs to be supported.

The following table describes the file format. The file should be placed in the in the %APPDATA%\Lockheed Martin\Prepar3D v5 folder, on the computer the server is running on.

XML	Values	Description
<SimBase.Document Type="SimConnect" version="1,0"></SimBase.Document>	String	SimConnect version information.
<Descr>SimConnect</Descr>	String	Description of this file.
<Filename>SimConnect.xml</Filename>	String	This filename.
<Disabled>False</Disabled>	True, False	Set to True to disable SimConnect completely.
<SimConnect.Comm></SimConnect.Comm>		Communications section. If remote connections are required, then additional SimConnect.Comm sections should be added, one for each protocol or scope that should be supported.
<Disabled>False</Disabled>	True, False	Set to True to disable this communication section.

<Protocol>/Pv6</Protocol>	Pipe (or Auto), IPv4, IPv6	The entry Auto is accepted for backwards compatibility as synonymous with Pipe. The protocol entered here is the only one this particular server will accept.
<Scope>/local</Scope>	local, global, link-local, unrestricted	For Pipe or IPv4, one of local or global. For IPv6, one of local, link-local, global, or unrestricted. Link-local is an IPv6 mechanism for accessing computers on a network that does not involve traversing a router. Unrestricted enables Teredo tunneling. Refer to Named-Pipe, IPv4 and IPv6 documentation on MSDN for more details.
<Address></Address> <Port></Port>		The server address and port. These should be copied to the identically named fields in the SimConnect.cfg files for the clients.
<MaxClients></MaxClients>	Integer	Define the maximum number of SimConnect clients that can be active at any one time, using this communication section.
<MaxRecvSize></MaxRecvSize>	Integer	The maximum receive packet size, in bytes. The server will disconnect a client that transmits a packet larger than this.
<DisableNagle>True</DisableNagle>	True, False	Set to True to disable the Nagle packet sending algorithms.

[- top -](#)

PREPAR3D

SimConnect.ini

```
[SimConnect]
level=verbose
console=1
;RedirectStdOutToConsole=1
;OutputDebugString=1
;file=%USERPROFILE%\Documents\Prepar3D v5 Files\SimConnect%03u.log
;file_next_index=0
;file_max_index=9
```

Notes:

- *The above example must be modified to correctly point to the current user's Documents directory.*
- *The file location should have write permissions for the current user.*

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

SimConnect.cfg

```
; Example SimConnect client configurations

[SimConnect]
Protocol=Auto
Address=
Port=
MaxReceiveSize=
DisableNagle=

[SimConnect.1]
Protocol=Pipe
Address=.

[SimConnect.2]
Protocol=Ipv6
Address=:1

[SimConnect.3]
Protocol=Ipv4
Address=127.0.0.1

[SimConnect.4]
Protocol=Pipe
Address=<remote computer address or name here>
Port=<remote computer pipe name (matches Port name given in SimConnect.xml)>

[SimConnect.5]
Protocol=IPv6
Address=<remote computer address or name here>
Port=<remote computer port number here>

[SimConnect.6]
Protocol=IPv4
Address=<remote computer address or name here>
Port=<remote computer port number here>
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

SimConnect.xml

```
<?xml version="1.0" encoding="Windows-1252"?>

<SimBase.Document Type="SimConnect" version="1,0">
  <Descr>SimConnect Server Configuration</Descr>
  <Filename>SimConnect.xml</Filename>
  <Disabled>False</Disabled>

  <!-- Example Global (remote) Pipe Server Configuration-->
  <SimConnect.Comm>
    <Disabled>True</Disabled>
    <Protocol>Pipe</Protocol>
    <Scope>global</Scope>
    <MaxClients>64</MaxClients>
    <Port>REPLACE_WITH_PORT_NAME</Port>
  </SimConnect.Comm>

  <!-- Example Global (remote) IPv6 Server Configuration-->
  <SimConnect.Comm>
    <Disabled>True</Disabled>
    <Protocol>IPv6</Protocol>
    <Scope>global</Scope>
    <MaxClients>64</MaxClients>
    <Address>::</Address>
    <Port>REPLACE_WITH_PORT_NUMBER</Port>
  </SimConnect.Comm>

  <!-- Example Global (remote) IPv4 Server Configuration-->
  <SimConnect.Comm>
    <Disabled>True</Disabled>
    <Protocol>IPv4</Protocol>
    <Scope>global</Scope>
    <MaxClients>64</MaxClients>
    <Address>0.0.0.0</Address>
    <Port>REPLACE_WITH_PORT_NUMBER</Port>
  </SimConnect.Comm>

  <!-- Example Local Pipe Server Configuration-->
  <SimConnect.Comm>
    <Disabled>True</Disabled>
    <Protocol>Pipe</Protocol>
    <Scope>local</Scope>
    <MaxClients>64</MaxClients>
    <Port>REPLACE_WITH_PORT_NAME</Port>
  </SimConnect.Comm>

  <!-- Example Local IPv6 Server Configuration-->
  <SimConnect.Comm>
    <Disabled>True</Disabled>
    <Protocol>IPv6</Protocol>
    <Scope>local</Scope>
    <MaxClients>64</MaxClients>
    <Address>::1</Address>
    <Port>REPLACE_WITH_PORT_NUMBER</Port>
  </SimConnect.Comm>

  <!-- Example Local IPv4 Server Configuration-->
  <SimConnect.Comm>
    <Disabled>True</Disabled>
    <Protocol>IPv4</Protocol>
    <Scope>local</Scope>
    <MaxClients>64</MaxClients>
    <Address>127.0.0.1</Address>
    <Port>REPLACE_WITH_PORT_NUMBER</Port>
  </SimConnect.Comm>

</SimBase.Document>
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

ExternalSim

Please note that ExternalSim is no longer directly supported and no future enhancements are planned for this feature.

The capabilities found in the [SimObject API](#) can be considered the second iteration of ExternalSim and is the recommended path forward for external simulation development and integration.

The ExternalSim capabilities of Prepar3D give developers the ability to create their own external simulations that can be run inside of Prepar3D. An example ExternalSim project is included with the Prepar3D SDK. The example shows how to create a simple East/West/North/South object movement simulation that is used to control the 'Externally Simulated Flyable Object (E.S.F.O.)' vehicle. To experiment with the E.S.F.O., copy the entire ESFO folder:

SimConnect Samples\ExternalSim\ESFO

to Prepar3D's miscellaneous SimObjects folder:

SimObjects\Misc

The vehicle will be selectable from the Select Aircraft/Vehicle user interface, under Prepar3D, as the ExternalSim E.S.F.O..

The E.S.F.O. can be moved East/West through the aileron controls, North/South through the elevation controls, and the elevation is controlled through the throttle.



When creating external simulations, the object's configuration file should specify the following information, per variation:

Configuration Data	
ExternalSimID	The GUID that is referenced in the ExternalSim dll.
ExternalSimData	Specific data related to the ExternalSim. (Optional)
ExternalSimModule	The location of the ExternalSim dll. (Optional)

Additionally, inside the sim.cfg file, under **General**, the category should be **ExternalSim**.

NOTE: If ExternalSims are not appearing in your aircraft selection screen, confirm that the object is inside the ..\SimObjects\Misc folder and that your Prepar3D.cfg has **ExternalSim** listed under **Main** in the **User Objects** setting variable.

Specific implementation details are described in the [SimConnect](#) documentation. For more information on the External SimObject container, please read the [Sample SimObject Container using an External Sim](#) article.

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

External SimObject Example

The sample SimObject container is located at:

SDK\SimConnect Samples\ExternalSim\ESFO

This directory can be copied into the following Prepar3D subdirectory:

SimObjects\Misc

The Prepar3D E.S.F.O. (Externally Simulated Flying Object) should be available in the Select Vehicle dialog.

The E.S.F.O. SimObject uses a simple visual model of a UFO (the same UFO visual model that's available as a scenario object) with the [SimConnect ExternalSim Sample](#) as the simulation module.

The SimObject has a 2D cockpit panel only. The 2D cockpit consist of just a G1000 PFD gauge. It has no Virtual Cockpit and no interior model defined.

The SimObject defines all the radios as being available (Comm, Nav, etc).

There are two variations of the SimObject defined, one called Slow and one called Fast. The difference between these variations is the string they pass to initialize the External Sim module. The Slow variant has lower max velocity values and Fast variant has higher max velocity values.

Files provided in the sample container

File	Description
sim.cfg	This is the main configuration file for the container. It defines two [fltsim.n] sections, one for the Slow variant and one for the Fast variant. Note the ExternalSimID , ExternalSimData , and ExternalSimModule items in the [fltsim.n] sections and the category=ExternalSim in the [General] section.
model\model.cfg	This is the visual model configuration file. This SimObject only defines an external visual model, there is no internal visual model defined (so no Virtual Cockpit views are available).
model\gen_secret_ufo.mdl	This is the external visual model of a UFO (the same UFO visual model available as a Scenario Object).
Modules\ExternalSim.dll	This is a prebuilt version of the SimConnect ExternalSim Sample . Replace this file with updated versions as you modify the sample code.
panel\panel.cfg	This is the panel configuration file. A single 2D panel is defined containing just a G1000 PFD gauge.
texture\texture.cfg	This is the texture configuration file.
texture*.dds	These are the textures required by the UFO visual model.

[- top -](#)

Samples

Overview

Click on the Working Sample column to display the source file of a working sample. Many of the samples include the use of functions not listed in the Highlighted Functions column, this list just indicates those functions the sample is designed to specifically demonstrate.

Unmanaged code samples

The following samples are written in C++.

Sample	Description	Highlighted Functions	Highlighted Structures or Enumerations
AI Objects and Waypoints	Creates a number of AI objects, including an aircraft, a truck, a hot air balloon and a whale. A key press sends waypoints to the aircraft and truck.	SimConnect_SetDataOnSimObject SimConnect_AICreateSimulatedObject SimConnect_AICreateNonATCAircraft	SIMCONNECT_DATA_INITPOSITION SIMCONNECT_DATA_WAYPOINT SIMCONNECT_RECV_ASSIGNED_OBJECT_ID
AI Traffic	Creates a number of aircraft at Yakima Air Term/Mcallister airport and provides them with a flight plan to Spokane Intl. Sets up two parked aircraft, and a key press initiates them with the same flight plan.	SimConnect_AISetAircraftFlightPlan SimConnect_AICreateEnrouteATCAircraft SimConnect_AICreateParkedATCAircraft	SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE SIMCONNECT_RECV_ASSIGNED_OBJECT_ID
Camera System	Modifies camera and containing view/window.	SimConnect_CreateCameraDefinition SimConnect_CreateCameraInstance SimConnect_OpenView SimConnect_UndockView SimConnect_DockView SimConnect_SetCameraWindowPosition SimConnect_SetCameraWindowSize	
Change Vehicle	Changes the Vehicle that is being used in Prepar3D.	SimConnect_ChangeVehicle	
Client Event	Request notifications of a simulation event, the applying of the user aircraft's brakes in this case.	SimConnect_MapClientEventToSimEvent SimConnect_AddClientEventToNotificationGroup SimConnect_SetNotificationGroupPriority SimConnect_CallDispatch	SIMCONNECT_RECV_EVENT
Cockpit Camera	Banks the user's view camera left and right in response to certain key presses.	SimConnect_CameraSetRelative6DOF SimConnect_MapInputEventToClientEvent SimConnect_AddClientEventToNotificationGroup SimConnect_MapClientEventToSimEvent	SIMCONNECT_RECV_EXCEPTION
Data Harvester	Requests data on the user object and outputs it to a CSV file.	SimConnect_RequestDataOnSimObject SimConnect_MenuAddItem SimConnect_MenuDeleteItem	
DialogBoxMode	Displays a message box, then resets Dialog mode.	SimConnect_SetSystemState SimConnect_RequestSystemState	SIMCONNECT_RECV_SYSTEM_STATE
ExternalSim	Provides a simple example of how to write a SimConnect External Sim.	SimConnect_RegisterExternalSim SimConnect_UnregisterExternalSim SimConnect_SynchronousUnlock SimConnect_MapClientEventToSimEvent SimConnect_AddToDataDefinition	SIMCONNECT_RECV_EXTERNAL_SIM_CREATE SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED SIMCONNECT_RECV_EXTERNAL_SIM_EVENT
External Sim Container	Sample SimObject container using the Sample External Sim above. Provides a complete SimObject container that can be dropped into the SimObjects\Misc\ folder to test the Sample External Sim .		
FacilitiesData	Uses text menus to select the options for retrieving facilities data, such as airports, VORs, NDBs etc.	SimConnect_RequestFacilitiesList SimConnect_SubscribeToFacilities SimConnect_UnsubscribeToFacilities SimConnect_Text	SIMCONNECT_RECV_FACILITIES_LIST SIMCONNECT_TEXT_RESULT SIMCONNECT_TEXT_TYPE
Input Event	Applies the user aircraft brakes by pressing the Ctrl-Shift-U key combination.	SimConnect_MapInputEventToClientEvent SimConnect_SetInputGroupState	SIMCONNECT_RECV_EVENT
Joystick Input	Provides the input data from the joystick X and Y axis position, Z axis rotation, hat switch and slider. A key press selects each input option in turn.	SimConnect_SetInputGroupState SimConnect_MapClientEventToSimEvent SimConnect_AddClientEventToNotificationGroup SimConnect_MapInputEventToClientEvent	SIMCONNECT_RECV_EVENT

Menu Items	Adds and removes a menu item.	SimConnect_MenuAddItem SimConnect_MenuDeleteItem SimConnect_RemoveClientEvent SimConnect_AddClientEventToNotificationGroup	SIMCONNECT_RECV_EVENT
Mission Action	Interfaces with a scenario object XML file to provide some additional processing to scenario actions, and when scenarios are completed.	SimConnect_ExecuteMissionAction SimConnect_CompleteCustomMissionAction SimConnect_SubscribeToSystemEvent WaitForSingleObject	SIMCONNECT_RECV_CUSTOM_ACTION
No Callback	Similar to the Client Event sample, but written without a callback function.	SimConnect_GetNextDispatch	SIMCONNECT_RECV_EVENT
Open and Close	Simply opens and closes a connection with the SimConnect server.	SimConnect_Open SimConnect_Close	
Pause Monitor	Simply monitors the pause state of Prepar3D.		
Request Data	Requests a small selection of data on the user's aircraft, including latitude, longitude and altitude.	SimConnect_AddToDataDefinition SimConnect_RequestDataOnSimObjectType	SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_SIMOBJECT_DATA
Scenery Complexity and Shadow Flags	Requests either the current scenery complexity setting or the current shadow flag settings based on a user event.	SimConnect_RequestSceneryComplexity SimConnect_RequestShadowFlags	SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_SCENERY_COMPLEXITY SIMCONNECT_RECV_SHADOW_FLAGS
Reserved Key	Requests that one of the three keys specified be reserved for this client.	SimConnect_MapClientEventToSimEvent SimConnect_RequestReservedKey	SIMCONNECT_RECV_RESERVED_KEY
Send Event A	To be used in combination with Send Event B and Send Event C. Transmits custom events to the two other clients.	SimConnect_TransmitClientEvent SimConnect_MapClientEventToSimEvent SimConnect_AddClientEventToNotificationGroup SimConnect_SetNotificationGroupPriority	SIMCONNECT_RECV_EVENT
Send Event B	Receives custom events from Send Event A, at a higher priority than Send Event C.	SimConnect_MapClientEventToSimEvent SimConnect_AddClientEventToNotificationGroup SimConnect_SetNotificationGroupPriority	SIMCONNECT_RECV_EVENT
Send Event C	Receives custom events from Send Event A, at a lower priority than Send Event B.	SimConnect_MapClientEventToSimEvent SimConnect_AddClientEventToNotificationGroup SimConnect_SetNotificationGroupPriority	SIMCONNECT_RECV_EVENT
Set Data	When a certain key is pressed, the position of the user aircraft is changed.	SimConnect_SetDataOnSimObject SimConnect_AddToDataDefinition SimConnect_MapClientEventToSimEvent SimConnect_MapInputEventToClientEvent SimConnect_SetInputGroupState	SIMCONNECT_DATA_INITPOSITION
System Event	Notifies the client when a FlightLoaded event has occurred.	SimConnect_SubscribeToSystemEvent	SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_EVENT_FILENAME
Tagged Data	Requests the vertical speed and pitot heat switch settings on the user aircraft, but only when this data changes.	SimConnect_RequestDataOnSimObject SimConnect_AddToDataDefinition	SIMCONNECT_RECV_SIMOBJECT_DATA
TextMenu	Displays and clears a text menu.	SimConnect_Text	SIMCONNECT_TEXT_RESULT SIMCONNECT_TEXT_TYPE
Throttle Control	Controls the user aircraft throttle using two specified keys.	SimConnect_SetDataOnSimObject SimConnect_AddToDataDefinition SimConnect_MapClientEventToSimEvent SimConnect_MapInputEventToClientEvent SimConnect_SetInputGroupState	SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE
Tracking Errors	Stores information on each call to the server, so that any error caught by the server can easily be located by the client. Primarily useful for debugging rather than retail builds.	SimConnect_GetLastSentPacketID SimConnect_MapClientEventToSimEvent SimConnect_AddClientEventToNotificationGroup SimConnect_SetNotificationGroupPriority SimConnect_CallDispatch	SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_EXCEPTION
Variable Strings	Extracts three variable length strings from a structure.	SimConnect_RetrieveString SimConnect_AddToDataDefinition SimConnect_RequestDataOnSimObjectType	SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_EXCEPTION SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE
Weather Station	Requests weather data from the nearest weather station to the user aircraft, every 10 seconds.	SimConnect_WeatherRequestObservationAtNearestStation SimConnect_RequestDataOnSimObject	SIMCONNECT_RECV_SIMOBJECT_DATA SIMCONNECT_RECV_WEATHER_OBSERVATION
Windows Event	Uses a Windows Event to improve the waiting time performance of a client.	CreateEvent SimConnect_Open WaitForSingleObject	SIMCONNECT_RECV_EVENT

Managed code samples

The following samples are mostly written in C#, with one in VB.NET, and all use the managed layer for SimConnect.

Sample	Description	Highlighted Functions	Highlighted Structures or Enumerations
--------	-------------	-----------------------	--

	<p>Retrieves all the SimObjects within the maximum radius around the User's vehicle.</p>		
AI Destroyer	<p>Checking a box next to the SimObject's name then sends SetHealth messages across to the SimObject. This allows the killing and subsequent revival of SimObjects to be tested.</p>	None	None
AI Waypoints	<p>Sends an array of waypoints to an Extra 300S, and a fuel truck.</p>	SimConnect_AICreateSimulatedObject SimConnect_AICreateNonATCAircraft SimConnect_AddToDataDefinition SimConnect_SetDataOnSimObject	SIMCONNECT_DATA_WAYPOINT SIMCONNECT_RECV_ASSIGNED_OBJECT_ID
Change Vehicle	<p>Changes the Vehicle that is being used in Prepar3D.</p>	SimConnect_ChangeVehicle	
Client Event	<p>Requests Pilot heat switch and Flaps events.</p>	SimConnect_MapClientEventToSimEvent SimConnect_AddClientEventToNotificationGroup SimConnect_SetNotificationGroupPriority OnRecvEvent	SIMCONNECT_RECV SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_EXCEPTION
Data Request	<p>Click a button to request the title, latitude, longitude and altitude of the user aircraft.</p>	SimConnect_AddToDataDefinition RegisterDataDefineStruct OnRecvSimobjectDataBytype SimConnect_RequestDataOnSimObjectType	SIMCONNECT_RECV_SIMOBJECT_DATA SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE
Facilities Request	<p>Provides buttons to request facility data for Airports, Waypoints, NDBs and VORs.</p>	SimConnect_SubscribeToFacilities SimConnect_UnsubscribeToFacilities	SIMCONNECT_RECV_FACILITIES_LIST
Mission Objects	<p>Shows an example using Windows Presentation Foundation of requesting scenario object information for goals, mission objectives, and flight segments. Additionally shows an example of how to resolve a goal.</p>	SimConnect_SetSystemEventState SimConnect_SubscribeToSystemEvent SimConnect_RequestGoalCount SimConnect_RequestMissionObjectiveCount SimConnect_RequestFlightSegmentCount SimConnect_RequestGoalDataByGUID SimConnect_RequestGoalDataByIndex SimConnect_RequestMissionObjectiveDataByGUID SimConnect_RequestMissionObjectiveDataByIndex SimConnect_RequestFlightSegmentDataByGUID SimConnect_RequestFlightSegmentDataByIndex SimConnect_ResolveGoal WPF Databinding WPF Message Loop	SIMCONNECT_RECV SIMCONNECT_EXCEPTION SIMCONNECT_RECV_FLIGHT_SEGMENT SIMCONNECT_RECV_GOAL SIMCONNECT_RECV_FLIGHT_SEGMENT_READY_FOR_GRADING SIMCONNECT_RECV_MISSION_OBJECT_COUNT SIMCONNECT_RECV_MISSION_OBJECTIVE SIMCONNECT_RECV_PARAMETER_RANGE SIMCONNECT_GOAL_RESOLUTION
Observer Control	<p>Creates an observer, updates an observer and request an observer's data.</p>	SimConnect_CreateObserver SimConnect_RequestObserverData	SIMCONNECT_DATA_OBSERVER SIMCONNECT_RECV_OBSERVER_DATA
Scenario Controller	<p>Launches a scenario specified by a user through the console.</p>	SimConnect_SubscribeToSystemEvent SimConnect_UnsubscribeFromSystemEvent SimConnect_SetSystemEventState	SIMCONNECT_RECV SIMCONNECT_RECV_EVENT
System Event	<p>Requests 4 second and simulation state system notifications.</p>	SimConnect_SubscribeToSystemEvent SimConnect_UnsubscribeFromSystemEvent	SIMCONNECT_RECV SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_EXCEPTION
VB Data Request	<p>Click a button to request the title, latitude, longitude and altitude of the user aircraft.</p>	SimConnect_AddToDataDefinition RegisterDataDefineStruct OnRecvSimobjectDataBytype	SIMCONNECT_RECV_SIMOBJECT_DATA SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE
Weapon Station Selection	<p>Shows an example using Windows Presentation Foundation that utilizes databinding to allow the selection and moniyotinh of multiple hardpoints on a SimObject.</p>	SimConnect_SetSystemState SimConnect_AddToDataDefinition SimConnect_SetDataOnSimObject SimConnect_SubscribeToSystemEvent WPF Databinding WPF Message Loop	SIMCONNECT_RECV SIMCONNECT_RECV_EVENT SIMCONNECT_RECV_EXCEPTION

- top -

PREPAR3D

AIObjects.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect AI Objects and Waypoints sample  
//  
// Description:  
//           Adds Non ATC controlled simulation objects.  
//           With the default aircraft at Seatac (KSEA)- turn off the engine  
//           and watch the antics.  
//           Press z to create the objects  
//           Press x to load them with their waypoint lists  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include <WinDef.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE hSimConnect = NULL;  
  
DWORD   MooneyID    = SIMCONNECT_OBJECT_ID_USER;  
DWORD   TruckID     = SIMCONNECT_OBJECT_ID_USER;  
  
static enum EVENT_ID {  
    EVENT_SIM_START,  
    EVENT_Z,  
    EVENT_X,  
};  
  
static enum DATA_REQUEST_ID7 {  
    REQUEST_MOONEY,  
    REQUEST_TRUCK,  
    REQUEST_WHALE,  
};  
  
static enum GROUP_ID {  
    GROUP_ZX,  
};  
  
static enum INPUT_ID {  
    INPUT_ZX,  
};  
  
static enum DEFINITION_ID {  
    DEFINITION WAYPOINT,  
};  
  
// Set up flags so these operations only happen once  
static bool plansSent      = false;  
static bool objectsCreated  = false;  
  
void sendFlightPlans()  
{  
    HRESULT hr;  
    SIMCONNECT_DATA WAYPOINT wp[3];    // Mooney waypoint list  
    SIMCONNECT_DATA WAYPOINT ft[2];    // Truck waypoint list  
  
    // Mooney aircraft should fly in circles across the North end of the runway  
    wp[0].Flags      = SIMCONNECT WAYPOINT SPEED REQUESTED;  
    wp[0].Altitude   = 800;  
    wp[0].Latitude   = 47 + (27.79/60);  
    wp[0].Longitude  = -122 - (18.46/60);  
    wp[0].ktsSpeed   = 100;  
  
    wp[1].Flags      = SIMCONNECT WAYPOINT SPEED REQUESTED;  
    wp[1].Altitude   = 600;  
    wp[1].Latitude   = 47 + (27.79/60);  
    wp[1].Longitude  = -122 - (17.37/60);  
    wp[1].ktsSpeed   = 100;  
  
    wp[2].Flags      = SIMCONNECT WAYPOINT_WRAP_TO_FIRST | SIMCONNECT WAYPOINT SPEED REQUESTED;  
    wp[2].Altitude   = 800;  
    wp[2].Latitude   = 47 + (27.79/60);  
    wp[2].Longitude  = -122 - (19.92/60);  
    wp[2].ktsSpeed   = 100;  
  
    // Send the three waypoints to the Mooney  
    hr = SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION WAYPOINT, MooneyID, 0, ARRAYSIZE(wp), sizeof(wp[0]), wp);  
  
    // Truck goes down the runway  
    ft[0].Flags      = SIMCONNECT WAYPOINT SPEED REQUESTED;  
    ft[0].Altitude   = 433;  
    ft[0].Latitude   = 47 + (25.93/60);  
    ft[0].Longitude  = -122 - (18.46/60);  
    ft[0].ktsSpeed   = 75;
```

```

        ft[1].Flags      = SIMCONNECT_WAYPOINT_WRAP_TO_FIRST | SIMCONNECT_WAYPOINT_SPEED_REQUESTED;
        ft[1].Altitude   = 433;
        ft[1].Latitude   = 47 + (26.25/60);
        ft[1].Longitude  = -122 - (18.46/60);
        ft[1].ktsSpeed   = 55;

        // Send the two waypoints to the truck
        hr = SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION_WAYPOINT, TruckID, 0, ARRSIZE(ft), sizeof(ft[0]), ft);

    }

void setUpSimObjects()
{
    SIMCONNECT_DATA_INITPOSITION Init;
    HRESULT hr;

    // Initialize Mooney aircraft just in front of user aircraft
    // User aircraft is at 47 25.89 N, 122 18.48 W

    Init.Altitude     = 433.0;           // Altitude of Sea-tac is 433 feet
    Init.Latitude     = 47 + (25.91/60);   // Convert from 47 25.90 N
    Init.Longitude    = -122 - (18.48/60); // Convert from 122 18.48 W
    Init.Pitch        = 0.0;
    Init.Bank         = 0.0;
    Init.Heading      = 360.0;
    Init.OnGround     = 1;
    Init.Airspeed     = 1;

    hr = SimConnect_AICreateNonATCAircraft(hSimConnect, "Mooney Bravo", "N1001", Init, REQUEST_MOONEY);

    // Initialize truck just in front of user aircraft
    // User aircraft is at 47 25.89 N, 122 18.48 W

    Init.Altitude     = 433.0;           // Altitude of Sea-tac is 433 feet
    Init.Latitude     = 47 + (25.91/60);   // Convert from 47 25.90 N
    Init.Longitude    = -122 - (18.47/60); // Convert from 122 18.48 W
    Init.Pitch        = 0.0;
    Init.Bank         = 0.0;
    Init.Heading      = 360.0;
    Init.OnGround     = 1;
    Init.Airspeed     = 0;

    hr = SimConnect_AICreateSimulatedObject(hSimConnect, "VEH_jetTruck", Init, REQUEST_TRUCK);

    // Add a humpback whale

    Init.Altitude     = 433.0;           // Altitude of Sea-tac is 433 feet
    Init.Latitude     = 47 + (25.89/60);   // Convert from 47 25.89 N
    Init.Longitude    = -122 - (18.51/60); // Convert from 122 18.51 W
    Init.Pitch        = 0.0;
    Init.Bank         = 0.0;
    Init.Heading      = 0.0;
    Init.OnGround     = 1;
    Init.Airspeed     = 0;
    hr = SimConnect_AICreateSimulatedObject(hSimConnect, "Humpbackwhale", Init, REQUEST_WHALE);
}

void CALLBACK MyDispatchProcSO(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;

            switch(evt->uEventID)
            {
                case EVENT_SIM_START:
                    // Sim has started so turn the input events on
                    hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_ZX, SIMCONNECT_STATE_ON);

                    break;

                case EVENT_Z:
                    if (!objectsCreated)
                    {
                        setUpSimObjects();
                        objectsCreated = true;
                    }
                    break;

                case EVENT_X:
                    if (!plansSent && objectsCreated)
                    {
                        sendFlightPlans();
                        plansSent = true;
                    }
                    break;

                default:
                    printf("\nUnknown event: %d", evt->uEventID);
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_ASSIGNED_OBJECT_ID:
        {

```

```

SIMCONNECT_RECV_ASSIGNED_OBJECT_ID *pObjData = (SIMCONNECT_RECV_ASSIGNED_OBJECT_ID*)pData;
switch( pObjData ->dwRequestID)
{
    case REQUEST_MOONEY:
        MooneyID = pObjData->dwObjectID;
        printf("\nCreated Mooney Bravo id = %d", MooneyID);
        break;

    case REQUEST_TRUCK:
        TruckID = pObjData->dwObjectID;
        printf("\nCreated truck id = %d", TruckID);
        break;

    case REQUEST_WHALE:
        printf("\nCreated humpback whale id = %d", pObjData->dwObjectID);
        break;

    default:
        printf("\nUnknown creation %d", pObjData->dwRequestID);
        break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    quit = 1;
    break;
}

default:
    printf("\nReceived:%d", pData->dwID);
    break;
}

void testSimObjects()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "AI Objects and Waypoints", NULL, 0, 0, 0)))
    {
        printf("\nConnected!");

        // Create some private events
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_Z);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_X);

        // Link the private events to keyboard keys, and ensure the input events are off
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_ZX, "Z", EVENT_Z);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_ZX, "X", EVENT_X);

        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_ZX, SIMCONNECT_STATE_OFF);

        // Sign up for notifications
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_ZX, EVENT_Z);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_ZX, EVENT_X);

        // Set up a definition for a waypoint list
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION WAYPOINT,
            "AI Waypoint List", "number", SIMCONNECT_DATATYPE WAYPOINT);

        // Request a simulation start event
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcSO, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testSimObjects();
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

AITraffic.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect AI ATC Aircraft sample  
//  
// Description:  
// Adds AI aircraft to make the flight from Yakima to Spokane busy.  
// First start the user aircraft at Yakima (or load the Yakima to Spokane  
// flight plan used by the AI aircraft - then drive off the runway to view  
// the goings on).  
// Press the Z key to add six AI aircraft  
// Press the X key to give the parked aircraft the Yakima to Spokane  
// flight plan  
// Both keys can only work once.  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int quit = 0;  
HANDLE hSimConnect = NULL;  
DWORD ParkedMauleID = SIMCONNECT_OBJECT_ID_USER;  
DWORD ParkedMooneyID = SIMCONNECT_OBJECT_ID_USER;  
  
static enum EVENT_ID {  
    EVENT_SIM_START,  
    EVENT_Z,  
    EVENT_X,  
    EVENT_ADDED_AIRCRAFT,  
    EVENT_REMOVED_AIRCRAFT,  
};  
  
static enum DATA_REQUEST_ID7 {  
    REQUEST_MOONEY1,  
    REQUEST_MAULE1,  
    REQUEST_MOONEY2,  
    REQUEST_MAULE2,  
    REQUEST_MAULE_PARKED,  
    REQUEST_MOONEY_PARKED,  
    REQUEST_MAULE_PARKED_FLIGHTPLAN,  
    REQUEST_MOONEY_PARKED_FLIGHTPLAN,  
};  
  
static enum GROUP_ID {  
    GROUP_ZX,  
};  
  
static enum INPUT_ID {  
    INPUT_ZX,  
};  
  
// Set up flags so these operations only happen once  
static bool plansSent = false;  
static bool aircraftCreated = false;  
  
void sendFlightPlans()  
{  
    HRESULT hr;  
  
    if (ParkedMauleID != SIMCONNECT_OBJECT_ID_USER)  
    {  
        hr = SimConnect_AISetAircraftFlightPlan(hSimConnect, ParkedMauleID,  
            "IFR Yakima Air Term Mcallister to Spokane Intl", REQUEST_MAULE_PARKED_FLIGHTPLAN);  
    }  
    if (ParkedMooneyID != SIMCONNECT_OBJECT_ID_USER)  
    {  
        hr = SimConnect_AISetAircraftFlightPlan(hSimConnect, ParkedMooneyID,  
            "IFR Yakima Air Term Mcallister to Spokane Intl", REQUEST_MOONEY_PARKED_FLIGHTPLAN);  
    }  
}  
  
void setUpAIACraft()  
{  
    HRESULT hr;  
  
    // Add some AI controlled aircraft  
    hr = SimConnect_AICreateEnrouteATCAircraft(hSimConnect, "Mooney Bravo Retro", "N100", 100,  
        "IFR Yakima Air Term Mcallister to Spokane Intl", 0.0f, false, REQUEST_MOONEY1);  
}
```

```

hr = SimConnect_AICreateEnrouteATCAircraft(hSimConnect, "Maule M7 260C paint4", "N101", 101,
    "IFR Yakima Air Term Mcallister to Spokane Intl", 0.0f, false, REQUEST_MAULE1);

hr = SimConnect_AICreateEnrouteATCAircraft(hSimConnect, "Mooney Bravo", "N200", 200,
    "IFR Yakima Air Term Mcallister to Spokane Intl", 0.0f, false, REQUEST_MOONEY2);

hr = SimConnect_AICreateEnrouteATCAircraft(hSimConnect, "Maule M7 260C paint4", "N201", 201,
    "IFR Yakima Air Term Mcallister to Spokane Intl", 0.0f, false, REQUEST_MAULE2);

// Park a few aircraft
hr = SimConnect_AICreateParkedATCAircraft(hSimConnect, "Maule M7 260C paint3", "N102",
    "KYKM", REQUEST_MAULE_PARKED);

hr = SimConnect_AICreateParkedATCAircraft(hSimConnect, "Mooney Bravo", "N202",
    "KYKM", REQUEST_MOONEY_PARKED);
}

void CALLBACK MyDispatchProcAI(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;

            switch(evt->uEventID)
            {
                case EVENT_SIM_START:

                    // Sim has started so turn input events on
                    hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_ZX, SIMCONNECT_STATE_ON);
                    break;

                case EVENT_Z:
                    if (!aircraftCreated)
                    {
                        setUpAAIAircraft();
                        aircraftCreated = true;
                    }
                    break;

                case EVENT_X:
                    if (!plansSent && aircraftCreated)
                    {
                        sendFlightPlans();
                        plansSent = true;
                    }
                    break;

                default:
                    printf("\nUnknown event: %d", evt->uEventID);
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE:
        {
            SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE *evt = (SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE*)pData;

            switch(evt->uEventID)
            {
                case EVENT_ADDED_AIRCRAFT:
                    printf("\nAI object added: Type=%d, ObjectID=%d", evt->eObjType, evt->dwData);
                    break;

                case EVENT_REMOVED_AIRCRAFT:
                    printf("\nAI object removed: Type=%d, ObjectID=%d", evt->eObjType, evt->dwData);
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_ASSIGNED_OBJECT_ID:
        {
            SIMCONNECT_RECV_ASSIGNED_OBJECT_ID *pObjData = (SIMCONNECT_RECV_ASSIGNED_OBJECT_ID*)pData;

            switch( pObjData ->dwRequestID)
            {
                // Do nothing specific in these cases, as the aircraft already have their flight plans

                case REQUEST_MOONEY1:
                    printf("\nCreated Mooney Bravo id = %d", pObjData->dwObjectID);
                    break;

                case REQUEST_MAULE1:
                    printf("\nCreated Maule M7 id = %d", pObjData->dwObjectID);
                    break;

                case REQUEST_MOONEY2:
                    printf("\nCreated Mooney Bravo id = %d", pObjData->dwObjectID);
                    break;

                case REQUEST_MAULE2:
                    printf("\nCreated Maule M7 id = %d", pObjData->dwObjectID);
                    break;
            }
        }
    }
}

```

```

    case REQUEST_MAULE_PARKED:
        // Record the object ID, so the flightplan can be sent out later
        ParkedMauleID = pObjData ->dwObjectID;
        printf("\nCreated parked Maule M7 %d", pObjData->dwObjectID);
        break;

    case REQUEST_MOONEY_PARKED:
        // Record the object ID, so the flightplan can be sent out later
        ParkedMooneyID = pObjData ->dwObjectID;
        printf("\nCreated parked Mooney %d", pObjData->dwObjectID);
        break;

    default:
        printf("\nUnknown creation %d", pObjData->dwRequestID);
        break;
    }
    break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    quit = 1;
    break;
}

default:
    printf("\nReceived:%d", pData->dwID);
    break;
}
}

void testAIAircraft()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "AI Traffic", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Create some private events
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_Z);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_X);

        // Link the private events to keyboard keys, and ensure input events are off
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_ZX, "Z", EVENT_Z);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_ZX, "X", EVENT_X);

        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_ZX, SIMCONNECT_STATE_OFF);

        // Sign up for notifications
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_ZX, EVENT_Z);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_ZX, EVENT_X);

        // Request a simulation start event
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

        // Subscribe to system events notifying the client that objects have been added or removed
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_ADDED_AIRCRAFT, "ObjectAdded");
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_REMOVED_AIRCRAFT, "ObjectRemoved");

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcAI, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testAIAircraft();
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

CameraSystem.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Camera System Sample
//
// Description:
//   Press the 'E' key to create a camera through SimConnect.
//   After the camera is created use the 'J', 'K', ',', and '.'
//   keys to modify the camera.
//
//   'J' Undocks the view
//   'K' Docks the view
//   ',' Changes the view's position
//   '.' Changes the view's size
//
// These controls are also printed to the console when the sample
// application starts.
//
//-----
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

// SimConnect globals
static bool G_QUIT = false;
static HANDLE hSimConnect = NULL;
static const int SIMCONNECT_SLEEP_TIME = 1;

// Sample project globals
static const char* PROJECT_NAME = "Camera System";
static const char* CONTROLS = "\n\n\nCamera System Sample\n\nControls:\n'E' - Create Camera\n'J' - Undock view \n'K' -\nDock view\n',' - Toggle view position\n'.' - Toggle view size\n";
static const char* CAMERA_NAME = "SimconnectCamera";

// Used to change window position and size values
static const int X1 = 200;
static const int Y1 = 200;
static const int X2 = 50;
static const int Y2 = 50;
static const int W1 = 512;
static const int H1 = 256;
static const int W2 = 256;
static const int H2 = 512;

// Keep track of current states
static bool WINDOW_POS_TOGGLE = true;
static bool WINDOW_SIZE_TOGGLE = true;
static bool CAMERA_CREATED = false;

static enum GROUP_ID
{
    GROUP0
};

static enum EVENT_ID
{
    EVENT_CAMERA_CREATE,
    EVENT_CAMERA_UNDOCK,
    EVENT_CAMERA_DOCK,
    EVENT_CAMERA_WINDOW_POS,
    EVENT_CAMERA_WINDOW_SIZE
};

static enum DATA_REQUEST_ID
{
    REQUEST_ID_0
};

HRESULT CreateCamera()
{
    HRESULT hr;

    SIMCONNECT_DATA_XYZ position;
    position.x = 0;
    position.y = 0;
    position.z = 0;

    SIMCONNECT_DATA_PBH rotation;
    rotation.Pitch = 0;
    rotation.Bank = 0;
    rotation.Heading = 0;

    GUID guid;
    CoCreateGuid(&guid);
}
```



```

        }

        break;

    case EVENT_CAMERA.Dock:
        if (CAMERA_CREATED)
        {
            hr = SimConnect_DockView(hSimConnect, CAMERA_NAME);
            if (hr == S_OK)
            {
                printf("\nView Docked");
            }
            else
            {
                printf("\nSimConnect error - Dock Camera");
            }
        }
        else
        {
            printf("\nCreate camera first! (press 'E')");
        }

        break;

    case EVENT_CAMERA_WINDOW_POS:
        if (CAMERA_CREATED)
        {
            int x;
            int y;
            GetWindowPos(x, y);
            hr = SimConnect_SetCameraWindowPosition(hSimConnect, CAMERA_NAME, x, y);
            if (hr == S_OK)
            {
                printf("\nWindow position updated");
            }
            else
            {
                printf("\nSimConnect error - Window Position");
            }
        }
        else
        {
            printf("\nCreate camera first! (press 'E')");
        }

        break;

    case EVENT_CAMERA_WINDOW_SIZE:
        if (CAMERA_CREATED)
        {
            int width;
            int height;
            GetWindowSize(width, height);
            hr = SimConnect_SetCameraWindowSize(hSimConnect, CAMERA_NAME, width, height);
            if (hr == S_OK)
            {
                printf("\nWindow size updated");
            }
            else
            {
                printf("\nSimConnect error - Window Size");
            }
        }
        else
        {
            printf("\nCreate camera first! (press 'E')");
        }

        break;

    default:
        break;
    }

    break;
}

case SIMCONNECT_RECV_ID_EXCEPTION:
{
    SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;

    switch (except->dwException)
    {
        case SIMCONNECT_EXCEPTION_ERROR:
            printf("\nCamera error");
            break;

        default:
            printf("\nException");
            break;
    }
    break;
}

case SIMCONNECT_RECV_ID_QUIT:
{

```

```

        G_QUIT = true;
        break;
    }

    default:
        break;
}
}

void TestCameraSystem()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, PROJECT_NAME, NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Define private events
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CAMERA_CREATE);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CAMERA_UNDOCK);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CAMERA_DOCK);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CAMERA_WINDOW_POS);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CAMERA_WINDOW_SIZE);

        // Group inputs
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_CAMERA_CREATE);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_CAMERA_UNDOCK);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_CAMERA_DOCK);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_CAMERA_WINDOW_POS);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_CAMERA_WINDOW_SIZE);

        // Set priorities
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

        // Map the keys , and . keys to the private events
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, GROUP0, "E", EVENT_CAMERA_CREATE);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, GROUP0, "J", EVENT_CAMERA_UNDOCK);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, GROUP0, "K", EVENT_CAMERA_DOCK);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, GROUP0, "VK_COMMA", EVENT_CAMERA_WINDOW_POS);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, GROUP0, "VK_PERIOD", EVENT_CAMERA_WINDOW_SIZE);

        // Set group 0 to be active on startup
        hr = SimConnect_SetInputGroupState(hSimConnect, GROUP0, SIMCONNECT_STATE_ON);

        // Show controls to user
        printf(CONTROLS);

        while (!G_QUIT)
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcCC, NULL);
            Sleep(SIMCONNECT_SLEEP_TIME);
        }

        hr = SimConnect_Close(hSimConnect);
    }
    else
    {
        printf("\nFailed to Connect");
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    TestCameraSystem();

    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

ChangeVehicle.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
//  SimConnect Change Vehicle sample  
//  
//  
//  Description:  Sample code for changing the Vehicle Prepar3D is running  
//                 through SimConnect.  
//  
//-----  
  
#include <Windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
HANDLE hSimConnect = NULL;  
  
void ChangeVehicle()  
{  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Change Vehicle", NULL, 0, 0, 0)))  
    {  
        // Sending the title of the vehicle  
        SimConnect_ChangeVehicle(hSimConnect, "Maule M7 260C paint2");  
  
        hr = SimConnect_Close(hSimConnect);  
    }  
}  
  
int __cdecl _tmain(int argc, _TCHAR* argv[])  
{  
    ChangeVehicle();  
  
    return 0;  
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

ClientEvent.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
//  SimConnect Client Event Sample  
//  
//  Description:  
//      Respond to the user aircraft brakes  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE  hSimConnect = NULL;  
  
static enum GROUP_ID {  
    GROUP0,  
};  
  
static enum EVENT_ID {  
    EVENT_BRAKES,  
};  
  
void CALLBACK MyDispatchProc1(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT_BRAKES:  
                    printf("\nEvent brakes: %d", evt->dwData);  
                    break;  
  
                default:  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_QUIT:  
        {  
            quit = 1;  
            break;  
        }  
  
        default:  
            break;  
    }  
}  
  
void testClientEvents()  
{  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Client Event", NULL, 0, NULL, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES, "brakes");  
  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_BRAKES);  
  
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);  
  
        while( 0 == quit )  
        {  
            SimConnect_CallDispatch(hSimConnect, MyDispatchProc1, NULL);  
            Sleep(1);  
        }  
  
        hr = SimConnect_Close(hSimConnect);  
    }  
    else  
        printf("\nFailed to Connect");  
}
```

```
int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testClientEvents();

    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

CockpitCamera.cpp

```

//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----[SimConnect Cockpit Camera Sample]-----
// Description:
//           Press the < and > keys (actually the , and . keys, as there is
//           no use of the shift key) to rotate the pilot's view.
//-----[SimConnect Cockpit Camera Sample]-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

int      quit = 0;
HANDLE   hSimConnect = NULL;

static enum GROUP_ID {
    GROUP0,
};

static enum EVENT_ID {
    EVENT_CAMERA_RIGHT,
    EVENT_CAMERA_LEFT,
};
static float cameraBank = 0.0f;

float normalize180 (float v)
{
    while (v < -180.0f) v += 360.0f;
    while (v > 180.0f) v -= 360.0f;
    return v;
}

void CALLBACK MyDispatchProcCC(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;

            switch(evt->uEventID)
            {
                case EVENT_CAMERA_RIGHT:

                    cameraBank = normalize180( cameraBank + 5.0f);

                    hr = SimConnect_CameraSetRelative6DOF(hSimConnect, 0.0f, 0.0f, 0.0f,
                        SIMCONNECT_CAMERA_IGNORE_FIELD,SIMCONNECT_CAMERA_IGNORE_FIELD, cameraBank);

                    printf("\nCamera Bank = %f", cameraBank);
                    break;

                case EVENT_CAMERA_LEFT:

                    cameraBank = normalize180( cameraBank - 5.0f);

                    hr = SimConnect_CameraSetRelative6DOF(hSimConnect, 0.0f, 0.0f, 0.0f,
                        SIMCONNECT_CAMERA_IGNORE_FIELD,SIMCONNECT_CAMERA_IGNORE_FIELD, cameraBank);

                    printf("\nCamera Bank = %f", cameraBank);
                    break;

                default:
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_EXCEPTION:
        {
            SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;

            switch (except->dwException)
            {

```

```

        case SIMCONNECT_EXCEPTION_ERROR:
            printf("\nCamera error");
            break;

        default:
            printf("\nException");
            break;
    }
    break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    quit = 1;
    break;
}

default:
    break;
}
}

static enum INPUT_ID {
    INPUT0,
};

void testCockpitCamera()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Cockpit Camera", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Define private events
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CAMERA_RIGHT);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_CAMERA_LEFT);

        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_CAMERA_RIGHT);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_CAMERA_LEFT);

        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

        // Map the keys , and . keys to the private events
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "VK_PERIOD", EVENT_CAMERA_RIGHT);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "VK_COMMAS", EVENT_CAMERA_LEFT);

        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT0, SIMCONNECT_STATE_ON);

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcCC, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
    else
        printf("\nFailed to Connect");
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testCockpitCamera();

    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

DialogBoxMode.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Dialog Box Mode sample
// Description:
//           If the key combination U+Q is typed, a request is sent to set
//           Dialog Mode, and if it is successful, a message box is rendered, and then
//           Dialog Mode is turned off.
//-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

int     quit = 0;
HANDLE hSimConnect = NULL;

static enum GROUP_ID {
    GROUP0,
};

static enum EVENT_ID {
    EVENT0,
};

static enum INPUT_ID {
    INPUT0,
};

static enum REQUEST_ID {
    REQUEST0,
};

void CALLBACK MyDispatchProc(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;

            switch(evt->uEventID)
            {
                case EVENT0:
                    printf("\nEVENT0: %d", evt->dwData);

                    // Send a request to turn Dialog Mode on
                    hr = SimConnect_SetSystemState( hSimConnect, "DialogMode", 1, 0, NULL );

                    // Send a request to ask whether Dialog Mode is on
                    hr = SimConnect_RequestSystemState(hSimConnect, REQUEST0, "DialogMode");
                    break;

                default:
                    printf("\nSIMCONNECT_RECV_EVENT: 0x%08X 0x%08X 0x%X", evt->uEventID, evt->dwData, cbData);
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_SYSTEM_STATE:
        {
            SIMCONNECT_RECV_SYSTEM_STATE *pState = (SIMCONNECT_RECV_SYSTEM_STATE*)pData;

            switch(pState->dwRequestId)
            {
                case REQUEST0:

                    // If Dialog Mode is on, show a Message box
                    if ( pState->dwInteger != 0 )
                    {
                        MessageBox( NULL, TEXT("Test!"), TEXT("Dialog Mode is on"), MB_OK );
                    }

                    // Send a request to turn Dialog Mode off
                    hr = SimConnect_SetSystemState( hSimConnect, "DialogMode", 0, 0, NULL );
            }
        }
    }
}
```

```

        }
        break;
    }

    printf("\nSIMCONNECT_RECV_SYSTEM_STATE RequestID=%d dwInteger=%d fFloat=%f szString=\"%s\"", pState->dwRequestID, pState->dwInteger, pState->fFloat, &pState->szString[0]);
    break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    quit = 1;
    break;
}

case SIMCONNECT_RECV_ID_EXCEPTION:
{
    SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;
    printf("\n\n***** EXCEPTION= %d SendID=%d uOffset=%d cbData=%d\n", except->dwException, except->dwSendID, except->dwIndex, cbData);
    break;
}

default:
    printf("\nUNKNOWN DATA RECEIVED: pData=%p cbData=%d\n", pData, cbData);
    break;
}

//-----  

// main  

//-----  

int __cdecl main(int argc, char* argv[])
{
    HANDLE hEventHandle = ::CreateEvent(NULL, FALSE, FALSE, NULL);
    if(hEventHandle == NULL)
    {
        printf("Error: Event creation failed! Bailing");
        return 1;
    }

    if (FAILED(SimConnect_Open(&hSimConnect, "DialogBoxMode Sample", NULL, 0, hEventHandle, 0)))
    {
        printf("\nConnection to Prepar3D failed!");

    } else
    {
        printf("\nConnected to Prepar3D!");

        HRESULT hr;

        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT0);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT0);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "U+Q", EVENT0);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT0, SIMCONNECT_STATE_ON);

        while( 0 == quit && hr==S_OK && ::WaitForSingleObject(hEventHandle, INFINITE) == WAIT_OBJECT_0 )
        {
            hr = SimConnect_CallDispatch(hSimConnect, MyDispatchProc, NULL);
        }
        hr = SimConnect_Close(hSimConnect);
        CloseHandle(hEventHandle);
    }

    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

```

ExternalSim.cpp
// ExternalSim.cpp : Test the SimConnect External Sim APIs
//
#include "StdAfx.h"
#include "SimConnect.h"
#include "ExternalSim.h"

HANDLE g_hSimConnect = NULL;
VehicleList g_VehicleList;

#define EXTERNAL_SIM_CALLBACK_MASK (SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_CREATE | SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_DESTROY | SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_SIMULATE | SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_LOCATION_CHANGED | SIMCONNECT_EXTERNAL_SIM_CALLBACK_FLAG_EVENT)

void OnRecvExternalSimCreate(SIMCONNECT_RECV_EXTERNAL_SIM_CREATE *pCreate)
{
    VehicleListIterator iter = g_VehicleList.find(pCreate->dwObjectID);
    // check that this vehicle is already tracking this vehicle
    if (iter != g_VehicleList.end())
    {
        // create new vehicle data structure
        VehicleData *pVehData = new VehicleData;
        pVehData->dwObjectID = pCreate->dwObjectID;
        pVehData->dwSimVarCount = pCreate->dwExternalSimVarBase;
        pVehData->id = pCreate->dwObjectID;
        pVehData->dwSimVar = pCreate->dwExternalSimVarBase;
        pVehData->dwSimVarEnd = pCreate->dwObjectID + dwSimVarCount - 1;
        if (pCreate->dwExternalSimVarBase < dwSimVarCount)
        {
            scsdfn(pCreate->dwExternalSimVarBase, &sf, &sf, &sf, &sf, &sf, &sf);
            pVehData->dwMaxLeftRightVelocity = sf;
            pVehData->dwMaxUpDownVelocity = sf;
            pVehData->dwMaxPitchVelocity = sf;
        }
        pVehData->bReset = true;
        // add vehicle to list
        g_VehicleList[pCreate->dwObjectID] = pVehData;
    }
}

SimConnect_SynchronousUnblock(g_hSimConnect);

void OnRecvExternalSimDestroy(SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY *pDestroy)
{
    VehicleListIterator iter = g_VehicleList.find(pDestroy->dwObjectID);
    // check that this object ID is being tracked currently
    if (iter != g_VehicleList.end())
    {
        // release vehicle and remove from local list
        delete iter->second;
        g_VehicleList.erase(iter);
    }
}

SimConnect_SynchronousUnblock(g_hSimConnect);

void OnRecvExternalSimSimulate(SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE *pSimulate)
{
    VehicleListIterator iter = g_VehicleList.find(pSimulate->dwObjectID);
    // check that this object ID is being tracked currently
    if (iter != g_VehicleList.end())
    {
        // yes, we know this vehicle, simulate it
        VehicleData *pVehData = iter->second;
        pVehData->pData = (PerVehicleSimulate*)lpSimulate->lpData;
        pVehData->dwDeltaTime = 1;
        // lat, lon, alt
        pVehData->Lat = pData->Lat;
        pVehData->Lon = pData->Lon;
        pVehData->Alt = pData->Alt;
        pVehData->Pitch = pData->Pitch;
        pVehData->Bank = pData->Bank;
        pVehData->Heading = pData->Heading;
    }
    if (pSimulate->dwDeltaTime > 1.5)
    {
        pVehData->dwDeltaTime = 0;
    }

    if (pVehData->bReset)
    {
        // if reset requested, reset values
        pVehData->fFrontBack = 0;
        pVehData->fLeftRight = 0;
        pVehData->fUpDown = 0;
        pVehData->fLeftRightVelocity = 0;
        pVehData->fUpDownVelocity = 0;
        pVehData->fFrontBackVelocity = 0;
        pVehData->fPitch = 0;
        pVehData->fBank = 0;
        pVehData->fHeading = 0;
    }

    SimConnect_SetDataInSimObject(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, pSimulate->dwObjectID, 0, 0, sizeof(PerVehicleOutput), &pvo);
}

// else if (bReset == true), then simulate this vehicle

// process inputs
pVehData->fFrontBackVelocity = pVehData->fMaxLeftRightVelocity * pVehData->fFrontBack;
pVehData->fUpDownVelocity = pVehData->fMaxUpDownVelocity * pVehData->fUpDown;
pVehData->fPitch = 0;

// step time integrate
pvo.Lat += LDM_METERS_TO_RADIANS(pVehData->fFrontBackVelocity * pSimulate->dwDeltaTime);
pvo.Lon += LDM_METERS_TO_RADIANS(pVehData->fLeftRightVelocity * pSimulate->dwDeltaTime);
pvo.Alt += pVehData->fUpDownVelocity * pSimulate->dwDeltaTime;
if (pVehData->GroundAltOffset < pVehData->fUpDown)
{
    if (pVehData->GroundAltOffset < pVehData->fUpDown)
    {
        pVehData->GroundAltOffset = pVehData->fUpDown;
    }
}
else
{
    if (SurfaceType != WATER, keep from going below ground
    if (pVehData->GroundAltOffset < 0)
    {
        pVehData->GroundAltOffset = 0;
    }
}
pVehData->Lat = pVehData->GroundAlt + pVehData->GroundAltOffset;
pVehData->Bank = 0;
if (pVehData->fFrontBackVelocity != 0 || pVehData->fLeftRightVelocity != 0)
{
    pvo.Roll = atan2(pVehData->fLeftRightVelocity, pVehData->fFrontBackVelocity);
    while (pvo.Roll > 2.0 * M_PI)
    {
        pvo.Roll -= 2.0 * M_PI;
    }
    while (pvo.Roll < -2.0 * M_PI)
    {
        pvo.Roll += 2.0 * M_PI;
    }
}

pvo.BodyVelX = 0.0;
pvo.BodyVelY = pVehData->fUpDownVelocity;
pvo.BodyVelZ = pVehData->fFrontBackVelocity * pVehData->fLeftRightVelocity + (pVehData->fFrontBackVelocity * pVehData->fFrontBackVelocity);
pvo.WorldVelX = pVehData->fLeftRightVelocity;
pvo.WorldVelY = pVehData->fUpDownVelocity;
pvo.WorldVelZ = pVehData->fFrontBackVelocity;
pvo.VerticalSpeed = pVehData->fUpDownVelocity;

SimConnect_SetDataInSimObject(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, pSimulate->dwObjectID, 0, 0, sizeof(PerVehicleOutput), &pvo);

SimConnect_SynchronousUnblock(g_hSimConnect);

void OnRecvExternalSimLocationChanged(SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED *pLocationChanged)
{
    if (pLocationChanged->bXeroSpeed)
    {
        VehicleListIterator iter = g_VehicleList.find(pLocationChanged->dwObjectID);
        if (iter != g_VehicleList.end())
        {
            iter->second->bReset = true;
        }
    }
}

SimConnect_SynchronousUnblock(g_hSimConnect);

void OnRecvExternalSimEvent(SIMCONNECT_RECV_EXTERNAL_SIM_EVENT *pEvent)
{
    VehicleListIterator iter = g_VehicleList.find(pEvent->dwObjectID);
    // check that this object ID is being tracked currently
    if (iter != g_VehicleList.end())
    {
        VehicleData *pVehData = iter->second;
        switch (pEvent->dwEventID)
        {
            case MOVEMENT_CENTER_EVENT:
            {
                pVehData->fFrontBack = 0;
                pVehData->fLeftRight = 0;
                pVehData->fUpDown = 0;
            }
            break;
            case LEFTRIGHT_INCREMENT_EVENT:
            {
                pVehData->fLeftRight += 0.1;
                if (pVehData->fLeftRight > 1.0)
                {
                    pVehData->fLeftRight = 1.0;
                }
                pVehData->fLeftRight = 1.0;
            }
            break;
            case LEFTRIGHT_DECREMENT_EVENT:
            {
                pVehData->fLeftRight -= 0.1;
                if (pVehData->fLeftRight < -1.0)
                {
                    pVehData->fLeftRight = -1.0;
                }
                pVehData->fLeftRight = -1.0;
            }
            break;
            case LEFTRIGHT_SET_EVENT:
            {
                pVehData->fLeftRight = (1.0*pEvent->dwData) / 100000.0;
                if (pVehData->fLeftRight > 1.0)
                {
                    pVehData->fLeftRight = 1.0;
                }
                if (pVehData->fLeftRight < -1.0)
                {
                    pVehData->fLeftRight = -1.0;
                }
            }
            break;
            case FRONTBACK_INCREMENT_EVENT:
            {
                pVehData->fFrontBack += 0.1;
                if (pVehData->fFrontBack > 1.0)
                {
                    pVehData->fFrontBack = 1.0;
                }
            }
            break;
            case FRONTBACK_DECREMENT_EVENT:
            {
                pVehData->fFrontBack -= 0.1;
                if (pVehData->fFrontBack < -1.0)
                {
                    pVehData->fFrontBack = -1.0;
                }
            }
            break;
        }
    }
}

```

```

        {
            pVehData->fFrontBack == 0.1;
            if (pVehData->fFrontBack < -1.0)
            {
                pVehData->fFrontBack = -1.0;
            }
            else
                pVehData->fFrontBack = 1.0;
        }
        break;
    case FRONTBACK_SET_EVENT:
        {
            pVehData->fFrontBack = ((int)pEvent->dwData) / 100000.0;
            if (pVehData->fFrontBack > 1.0)
            {
                pVehData->fFrontBack = 1.0;
            }
            if (pVehData->fFrontBack < -1.0)
            {
                pVehData->fFrontBack = -1.0;
            }
        }
        break;
    case UPDOWN_MAX_EVENT:
        {
            pVehData->fUpDown = 1.0;
        }
        break;
    case UPDOWN_MIN_EVENT:
        {
            pVehData->fUpDown = -1.0;
        }
        break;
    case UPDOWN_INCREMENT_EVENT_1:
        {
            pVehData->fUpDown += 0.1;
        }
        break;
    case UPDOWN_INCREMENT_EVENT_2:
        {
            pVehData->fUpDown += 0.1;
        }
        break;
    case UPDOWN_INCREMENT_EVENT_3:
        {
            pVehData->fUpDown += 0.1;
            if (pVehData->fUpDown > 1.0)
            {
                pVehData->fUpDown = 1.0;
            }
        }
        break;
    case UPDOWN_DECREMENT_EVENT_1:
        {
            pVehData->fUpDown -= 0.1;
        }
        break;
    case UPDOWN_DECREMENT_EVENT_2:
        {
            pVehData->fUpDown -= 0.1;
        }
        break;
    case UPDOWN_DECREMENT_EVENT_3:
        {
            pVehData->fUpDown -= 0.1;
            if (pVehData->fUpDown < -1.0)
            {
                pVehData->fUpDown = -1.0;
            }
        }
        break;
    case UPDOWN_SET_EVENT:
        {
            pVehData->fUpDown = pEvent->dwData / 100000.0;
            if (pVehData->fUpDown > 1.0)
            {
                pVehData->fUpDown = 1.0;
            }
            if (pVehData->fUpDown < -1.0)
            {
                pVehData->fUpDown = -1.0;
            }
        }
        break;
    case LEFTRIGHT_AXIS_EVENT:
        {
            pVehData->fLeftRight = -(int)pEvent->dwData / 16384.0;
        }
        break;
    case FRONTBACK_AXIS_EVENT:
        {
            pVehData->fFrontBack = (int)pEvent->dwData / 16384.0;
        }
        break;
    case UPDOWN_AXIS_EVENT:
        {
            pVehData->fUpDown = (int)pEvent->dwData / 16384.0;
        }
        break;
}
}

SimConnect_SynchronousUnblock(g_hSimConnect);

void OnRecvException(SIMCONNECT_RECV_EXCEPTION *pExceptionData)
{
}

void OnRecvQuit()
{
    SimConnect_UnregisterExternalSim(g_hSimConnect, g_guidExternalSim);
}

void OnRecvOpen(SIMCONNECT_RECV_OPEN *pOpenData)
{
// this struct/requests used to get data via the External sim Simulate callback
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("PLANE LATITUDE"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("PLANE LONGITUDE"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("PLANE ALTITUDE"), ("feet"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("PLANE PITCH DEGREES"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("PLANE BANK DEGREES"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("GROUND ALTITUDE"), ("meters"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("GND SPEED"), ("meters"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_SIMULATE_DEFINITION, ("SURFACE TYPE"), ("enum"), SIMCONNECT_DATATYPE_FLOAT64);

// map this struct/request used to send data back to the sim from the External Sim Simulate callback
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("PLANE LATITUDE"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("PLANE LONGITUDE"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("PLANE ALTITUDE"), ("feet"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("PLANE PITCH DEGREES"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("PLANE BANK DEGREES"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("PLANE HEADING DEGREES TRUE"), ("radians"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("VELOCITY BODY Y"), ("m/s"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("VELOCITY BODY X"), ("m/s"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("VELOCITY WORLD X"), ("m/s"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("VELOCITY WORLD Y"), ("m/s"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("VELOCITY WORLD Z"), ("m/s"), SIMCONNECT_DATATYPE_FLOAT64);
SimConnect_AddFobTableDefinition(g_hSimConnect, PER_VEHICLE_OUTPUT_DEFINITION, ("VERTICAL SPEED"), ("m/s"), SIMCONNECT_DATATYPE_FLOAT64);

// register as an external sim client
SimConnect_RegisterExternalSim(g_hSimConnect, g_guidExternalSim, EXTERNAL_SIM_CALLBACK_MASK, PER_VEHICLE_SIMULATE_DEFINITION);

// map some events
SimConnect_MapClientEventToSimEvent(g_hSimConnect, MOVEMENT_CENTER_EVENT, ("CENTER_AXES_BUDGER"));

// map some events
SimConnect_MapClientEventToSimEvent(g_hSimConnect, LEFTRIGHT_INCREMENT_EVENT, ("AILERONS_RIGHT"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, LEFTRIGHT_DECREMENT_EVENT, ("AILERONS_LEFT"));
SimConnect_MapClientEventToSimEvent(LEFTRIGHT_SET_EVENT, ("AILERON_SET"));

SimConnect_MapClientEventToSimEvent(g_hSimConnect, FRONTBACK_INCREMENT_EVENT, ("ELEVATOR_DOWN"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, FRONTBACK_DECREMENT_EVENT, ("ELEVATOR_UP"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, FRONTBACK_SET_EVENT, ("ELEVATOR_SET"));

SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_MAX_EVENT, ("THROTTLE_FULL"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_MIN_EVENT, ("THROTTLE_CUT"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_INCREMENT_EVENT_1, ("THROTTLE_INCH"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_INCREMENT_EVENT_2, ("THROTTLE_INCH_SMALL"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_DECREMENT_EVENT_1, ("THROTTLE_DECN"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_DECREMENT_EVENT_2, ("THROTTLE_DECN_SMALL"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_SET_EVENT, ("THROTTLE_SET"));

SimConnect_MapClientEventToSimEvent(g_hSimConnect, LEFTRIGHT_AXIS_EVENT, ("AXIS_AILERONS_SET"));
SimConnect_MapClientEventToSimEvent(g_hSimConnect, UPDOWN_AXIS_EVENT, ("AXIS_THROTTLE_SET"));
}

void CALLBACK SimConnectProcess(SIMCONNECT_RECV *pData, DWORD cbData, void *pContext)
{
// process SIMCONNECT_RECV_ID_XXX values here as needed
switch(pData->dwID)
{
    case SIMCONNECT_RECV_ID_EXTERNAL_SIM_CREATE:
        {
            OnRecvExternalSimCreate((SIMCONNECT_RECV_EXTERNAL_SIM_CREATE*)pData);
        }
        break;

    case SIMCONNECT_RECV_ID_EXTERNAL_SIM_DESTROY:
        {
            OnRecvExternalSimDestroy((SIMCONNECT_RECV_EXTERNAL_SIM_DESTROY*)pData);
        }
        break;

    case SIMCONNECT_RECV_ID_EXTERNAL_SIM_SIMULATE:
        {
            OnRecvExternalSimSimulate((SIMCONNECT_RECV_EXTERNAL_SIM_SIMULATE*)pData);
        }
        break;

    case SIMCONNECT_RECV_ID_EXTERNAL_SIM_LOCATION_CHANGED:
        {
            OnRecvExternalSimLocationChanged((SIMCONNECT_RECV_EXTERNAL_SIM_LOCATION_CHANGED*)pData);
        }
        break;

    case SIMCONNECT_RECV_ID_EXTERNAL_SIM_EVENT:
        {
            OnRecvExternalSimEvent((SIMCONNECT_RECV_EXTERNAL_SIM_EVENT*)pData);
        }
        break;

    case SIMCONNECT_RECV_ID_EXCEPTION:
        {
            OnRecvException((SIMCONNECT_RECV_EXCEPTION*)pData);
        }
        break;

    case SIMCONNECT_RECV_ID_QUIT:
        {
            OnRecvQuit();
        }
        break;

    case SIMCONNECT_RECV_ID_OPEN:
        {
            OnRecvOpen((SIMCONNECT_RECV_OPEN*)pData);
        }
        break;
}
}

void __attribute__((DLLStart)) DLLStart(void)
{
// open connection to local SimConnect server
if (SUCCEEDED(SimConnect_Open(g_hSimConnect, ("ExternalSim"), NULL, 0, NULL, SIMCONNECT_OPEN_CONFIGINDEX_LOCAL)) && g_hSimConnect != NULL)
{
    // register callback routine for message processing
    SimConnect_CallDispatch(g_hSimConnect, SimConnectProcess, NULL);
}
}

void __attribute__((DLLStop)) DLLStop(void)
{
if (g_hSimConnect != NULL)
{
    SimConnect_Close(g_hSimConnect);
    g_hSimConnect = NULL;
}
q_VehicleList.clear();
}

```


PREPAR3D

```
FacilitiesData.cpp
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//
// SimConnect FacilitiesData sample
//
// Description:
//           Ctrl F1 displays the Get Facilities menu on the screen
//           Ctrl F2 displays the Subscribe to Facilities menu on the screen
//-----
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

enum GROUP_ID
{
    GROUP_0,
};

enum EVENT_ID
{
    EVENT_0,
    EVENT_1,
    EVENT_TEXT_1,
    EVENT_TEXT_2,
    EVENT_MENU_1,
    EVENT_MENU_2,
    EVENT_MENU_3,
    EVENT_MENU_4,
};

enum INPUT_ID
{
    INPUT_0,
};

enum REQUEST_ID
{
    REQUEST_0,
    REQUEST_1
};

#pragma pack(push, 1)
#pragma pack(pop)

bool gQuit = false;
HANDLE hSimConnect = NULL;

static const char szTitle[] = "Facilities Data";
static const char szHelp[] = "Press Ctrl-F1 for Get Facilities, Ctrl-F2 for Subscribe to Facilities";

static const char GetFacilitiesMenu[] = \
"SimConnect Facilities Test\0"
"Choose which item:\0"
"Get airport facilities\0"
"Get waypoints\0"
"Get NDB\0"
"Get VOR\0"
"Get Tacan\0"
"Close menu\0";

static const char SubscribeOptionsMenu[] = \
"SimConnect Facilities Test\0"
"Choose which item:\0"
"Subscribe Facilities...\0"
"Unsubscribe Facilities...\0"
"Close menu\0";

static const char SubscribeFacilitiesMenu[] = \
"SimConnect Facilities Test\0"
"Choose which item:\0"
"Subscribe to airports\0"
"Subscribe to waypoints \0"
"Subscribe to NDB\0"
"Subscribe to VOR\0"
"Subscribe to Tacan\0"
"Back\0"
"Close menu\0";

static const char UnsubscribeFacilitiesMenu[] = \
"SimConnect Facilities Test\0"
"Choose which item:\0"
"Unsubscribe to airports\0"
"Unsubscribe to waypoints \0"
"Unsubscribe to NDB\0"
"Unsubscribe to VOR\0"
"Unsubscribe to Tacan\0"
"Back\0"
"Close menu\0";

char const * MenuText(SIMCONNECT_TEXT_RESULT result)
{
    switch (result)
    {
        case SIMCONNECT_TEXT_RESULT_DISPLAYED:
            return "Displayed";
            break;
        case SIMCONNECT_TEXT_RESULT_QUEUED:
            return "Queued";
            break;
        case SIMCONNECT_TEXT_RESULT_REMOVED:
            return "Removed from Queue";
            break;
        case SIMCONNECT_TEXT_RESULT_REPLACEABLE:
            return "Replaced in Queue";
            break;
        case SIMCONNECT_TEXT_RESULT_TIMEOUT:
            return "Timeout";
            break;
        default:
            if (SIMCONNECT_TEXT_RESULT_MENU_SELECT_1 <= result && result <= SIMCONNECT_TEXT_RESULT_MENU_SELECT_10)
            {
                static char text[] = "10 Selected";
                text[1] = result - SIMCONNECT_TEXT_RESULT_MENU_SELECT_1 + '1';
                return text + (result != SIMCONNECT_TEXT_RESULT_MENU_SELECT_10);
            }
            return "<unknown>";
    }
}

// Dump facility list head
void Dump(SIMCONNECT_RECV_FACILITIES_LIST * pList)
{
    printf("RequestID: %d dwArraySize: %d dwEntryNumber: %d dwOutOf: %d",
          pList->dwRequestId, pList->dwArraySize, pList->dwEntryNumber, pList->dwOutOf);
}
```

```

// Dump various facility elements
void Dump(SIMCONNECT_DATA_FACILITY_AIRPORT * pFac)
{
    printf(" Icao: %s Latitude: %lg Longitude: %lg Altitude: %lg",
           pFac->Icao, pFac->Latitude, pFac->Longitude, pFac->Altitude);
}

void Dump(SIMCONNECT_DATA_FACILITY_WAYPOINT * pFac)
{
    Dump((SIMCONNECT_DATA_FACILITY_AIRPORT*) pFac);
    printf(" fMagVar: %g", pFac->fMagVar);
}

void Dump(SIMCONNECT_DATA_FACILITY_NDB * pFac)
{
    Dump((SIMCONNECT_DATA_FACILITY_WAYPOINT*) pFac);
    printf(" frequency: %d", pFac->frequency);
}

void Dump(SIMCONNECT_DATA_FACILITY_VOR * pFac)
{
    Dump((SIMCONNECT_DATA_FACILITY_NDB*) pFac);
    printf(" Flags: %x fLocalizer: %f GlideLat: %lg GlideLon: %lg GlideAlt: %lg fGlideSlopeAngle: %f",
           pFac->Flags, pFac->fLocalizer, pFac->GlideLat, pFac->GlideLon, pFac->GlideAlt, pFac->fGlideSlopeAngle);
}

template <class T>
void DumpAll(T * pList)
{
    printf("\n");
    Dump(pList);
    for (unsigned i = 0; i < pList->dwArraySize; ++i)
    {
        printf("\n");
        Dump(pList->rgData + i);
    }
}

template <class T>
void RespondTo(T * pList, SIMCONNECT_FACILITY_LIST_TYPE type)
{
    DumpAll(pList);
}

void CALLBACK MySignalProc(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    switch(pData->dwID)
    {
        HRESULT hr;

        case SIMCONNECT_RECV_ID_OPEN:
        {
            SIMCONNECT_RECV_OPEN *pOpen = (SIMCONNECT_RECV_OPEN*)pData;
            printf("open: AppName=\"%s\" AppVersion=%d.%d.%d.%d SimConnectVersion=%d.%d.%d.%d.%d.%d",
                   pOpen->szApplicationName,
                   pOpen->dwApplicationVersionMajor, pOpen->dwApplicationVersionMinor, pOpen->dwApplicationBuildMajor, pOpen->dwApplicationBuildMinor,
                   pOpen->dwSimConnectVersionMajor, pOpen->dwSimConnectVersionMinor, pOpen->dwSimConnectBuildMajor, pOpen->dwSimConnectBuildMinor
            );
            break;
        }

        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;
            switch (evt->uEventID)
            {
                case EVENT_0:
                {
                    printf("\nEVENT0: %d", evt->dwData);
                    SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_1, sizeof(GetFacilitiesMenu), (void*)GetFacilitiesMenu);
                    break;
                }

                case EVENT_1:
                {
                    printf("\nEVENT1: %d", evt->dwData);
                    SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_2, sizeof(SubscribeOptionsMenu), (void*)SubscribeOptionsMenu);
                    break;
                }

                case EVENT_MENU_1:
                {
                    printf("\nMENU #1 Event: dwData=%d (%s)", evt->dwData, MenuText(SIMCONNECT_TEXT_RESULT(evt->dwData)));
                    unsigned const item = evt->dwData - SIMCONNECT_TEXT_RESULT_MENU_SELECT_1;
                    if (item < SIMCONNECT_FACILITY_LIST_TYPE_COUNT)
                    {
                        // Get the current cached list of airports, waypoints, etc, as the item indicates
                        hr = SimConnect_RequestFacilitiesList(hSimConnect, SIMCONNECT_FACILITY_LIST_TYPE(item), REQUEST_0);
                    }
                    break;
                }

                case EVENT_MENU_2:
                {
                    printf("\nMENU #1 Event: dwData=%d (%s)", evt->dwData, MenuText(SIMCONNECT_TEXT_RESULT(evt->dwData)));
                    switch (evt->dwData)
                    {
                        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_1:
                        {
                            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_3, sizeof(SubscribeFacilitiesMenu), (void*)SubscribeFacilitiesMenu);
                            break;
                        }

                        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_2:
                        {
                            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_4, sizeof(UnsubscribeFacilitiesMenu), (void*)UnsubscribeFacilitiesMenu);
                            break;
                        }
                    }
                    break;
                }

                case EVENT_MENU_3:
                {
                    printf("\nMENU #3 Event: dwData=%d (%s)", evt->dwData, MenuText(SIMCONNECT_TEXT_RESULT(evt->dwData)));
                    unsigned const item = evt->dwData;
                    if (item < SIMCONNECT_FACILITY_LIST_TYPE_COUNT)
                    {
                        hr = SimConnect_SubscribeToFacilities(hSimConnect, SIMCONNECT_FACILITY_LIST_TYPE(item), REQUEST_1);
                    }
                    else if (item == SIMCONNECT_FACILITY_LIST_TYPE_COUNT) // Back
                    {
                        SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_2, sizeof(SubscribeOptionsMenu), (void*)SubscribeOptionsMenu);
                    }
                    break;
                }

                case EVENT_MENU_4:
                {
                    printf("\nMENU #4 Event: dwData=%d (%s)", evt->dwData, MenuText(SIMCONNECT_TEXT_RESULT(evt->dwData)));
                    unsigned const item = evt->dwData;
                    if (item < SIMCONNECT_FACILITY_LIST_TYPE_COUNT)
                    {
                        hr = SimConnect_UnsubscribeToFacilities(hSimConnect, SIMCONNECT_FACILITY_LIST_TYPE(item));
                    }
                    else if (item == SIMCONNECT_FACILITY_LIST_TYPE_COUNT) // Back
                    {
                        SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_2, sizeof(SubscribeOptionsMenu), (void*)SubscribeOptionsMenu);
                    }
                    break;
                }

                default:
                {
                    printf("\nSIMCONNECT_RECV_EVENT: 0x%08X 0x%08X 0x%X", evt->uEventID, evt->dwData, cbData);
                    break;
                }
            }
        }
    }
}

```

```

        break;
    }

    case SIMCONNECT_RECV_ID_AIRPORT_LIST:
    {
        RespondTo((SIMCONNECT_RECV_AIRPORT_LIST*)pData, SIMCONNECT_FACILITY_LIST_TYPE_AIRPORT);
        break;
    }

    case SIMCONNECT_RECV_ID_WAYPOINT_LIST:
    {
        RespondTo((SIMCONNECT_RECV_WAYPOINT_LIST*)pData, SIMCONNECT_FACILITY_LIST_TYPE_WAYPOINT);
        break;
    }

    case SIMCONNECT_RECV_ID_NDB_LIST:
    {
        RespondTo((SIMCONNECT_RECV_NDB_LIST*)pData, SIMCONNECT_FACILITY_LIST_TYPE_NDB);
        break;
    }

    case SIMCONNECT_RECV_ID_VOR_LIST:
    {
        RespondTo((SIMCONNECT_RECV_VOR_LIST*)pData, SIMCONNECT_FACILITY_LIST_TYPE_VOR);
        break;
    }

    case SIMCONNECT_RECV_ID_TACAN_LIST:
    {
        RespondTo((SIMCONNECT_RECV_TACAN_LIST*)pData, SIMCONNECT_FACILITY_LIST_TYPE_TACAN);
        break;
    }

    case SIMCONNECT_RECV_ID_QUIT:
    {
        printf("\n***** SIMCONNECT_RECV_ID_QUIT *****\n");
        gQuit = true;
        break;
    }

    case SIMCONNECT_RECV_ID_EXCEPTION:
    {
        SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;
        printf("\n***** EXCEPTION=%d SendID=%d uOffset=%d cbData=%d\n", except->dwException, except->dwSendID, except->dwIndex, cbData);
        break;
    }

    default:
        printf("\nUNKNOWN DATA RECEIVED: pData=%p cbData=%d\n", pData, cbData);
        break;
    }
}

//-----  

// main  

//-----  

int __cdecl main(int argc, char* argv[])
{
    HANDLE hEventHandle = ::CreateEvent(NULL, FALSE, FALSE, NULL);
    if(hEventHandle == NULL)
    {
        printf("Error: Event creation failed! Exiting");
        return GetLastError();
    }

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "FacilitiesData", NULL, 0, hEventHandle, 0)))
    {
        printf("\nConnected to Prepar3D!");

        HRESULT hr;

        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_0);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1);

        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_0, TRUE);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_1, TRUE);
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_0, "Ctrl+F1", EVENT_0);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_0, "Ctrl+F2", EVENT_1);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_0, SIMCONNECT_STATE_ON);

        SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_PRINT_RED, 15, EVENT_TEXT_1, sizeof(szTitle), (void*)szTitle);
        SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_PRINT_RED, 15, EVENT_TEXT_2, sizeof(szHelp), (void*)szHelp);

        while (!gQuit && ::WaitForSingleObject(hEventHandle, INFINITE) == WAIT_OBJECT_0)
        {
            hr = SimConnect_CallDispatch(hSimConnect, MySignalProc, NULL);
            if (FAILED(hr))
            {
                break;
            }
        }

        hr = SimConnect_Close(hSimConnect);
        CloseHandle(hEventHandle);
    }
    else
        printf("\nFailed to Connect");
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

InputEvent.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect Input Sample  
//  
// Description:  
//           Ctrl-Shift-U key combination sets the brakes  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int      quit = 0;  
HANDLE   hSimConnect = NULL;  
  
static enum GROUP_ID {  
    GROUP0,  
};  
  
static enum EVENT_ID {  
    EVENT_BRAKES,  
};  
  
void CALLBACK MyDispatchProc1(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT_BRAKES:  
                    printf("\nEvent brakes: %d", evt->dwData);  
                    break;  
  
                default:  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_QUIT:  
        {  
            quit = 1;  
            break;  
        }  
  
        default:  
            break;  
    }  
}  
  
static enum INPUT_ID {  
    INPUT0,  
};  
  
void testInputEvents()  
{  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Input Event", NULL, 0, 0, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES, "brakes");  
  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT_BRAKES);  
  
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);  
  
        // Note that this does not override "." for brakes - both will be transmitted  
  
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "shift+ctrl+u", EVENT_BRAKES);  
  
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT0, SIMCONNECT_STATE_ON);  
    }
```

```
    while( 0 == quit )
    {
        SimConnect_CallDispatch(hSimConnect, MyDispatchProc1, NULL);
        Sleep(1);
    }

    hr = SimConnect_Close(hSimConnect);
}
else
    printf("\nFailed to Connect");
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testInputEvents();
    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

JoystickInput.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
//  SimConnect Joystick Control Sample  
//  
//  Description:  
//      Use the "z" key to step through the events sent by the Joystick  
//      including X,Y,Z axes, Slider and Hat switch  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE hSimConnect = NULL;  
int     current = 0;  
  
static enum GROUP_ID {  
    GROUP_0  
};  
  
static enum INPUT_ID {  
    INPUT_Z,  
    INPUT_SLIDER,  
    INPUT_XAXIS,  
    INPUT_YAXIS,  
    INPUT_RZAXIS,  
    INPUT_HAT,  
};  
  
static enum EVENT_ID {  
    EVENT_Z,  
    EVENT_SLIDER,  
    EVENT_XAXIS,  
    EVENT_YAXIS,  
    EVENT_RZAXIS,  
    EVENT_HAT,  
};  
  
void CALLBACK MyDispatchProcJ(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    HRESULT hr;  
  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT_SLIDER:  
                    printf("\nSlider value:%d", evt->dwData);  
                    break;  
                case EVENT_XAXIS:  
                    printf("\nX Axis value:%d", evt->dwData);  
                    break;  
                case EVENT_YAXIS:  
                    printf("\nY Axis value:%d", evt->dwData);  
                    break;  
                case EVENT_RZAXIS:  
                    printf("\nRotate Z axis value:%d", evt->dwData);  
                    break;  
                case EVENT_HAT:  
                    printf("\nHat value:%d", evt->dwData);  
                    break;  
  
                case EVENT_Z:  
                    current++;  
                    if (current == 6)  
                        current = 1;  
                    switch( current )  
                    {  
                        case 1:  
                            printf("\nSLIDER is active");  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_SLIDER, SIMCONNECT_STATE_ON);  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_XAXIS, SIMCONNECT_STATE_OFF);  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_YAXIS, SIMCONNECT_STATE_OFF);  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_RZAXIS, SIMCONNECT_STATE_OFF);  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_HAT, SIMCONNECT_STATE_OFF);  
                            break;  
                        case 2:  
                            printf("\nXAXIS is active");  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_XAXIS, SIMCONNECT_STATE_ON);  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_YAXIS, SIMCONNECT_STATE_OFF);  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_RZAXIS, SIMCONNECT_STATE_OFF);  
                            hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_HAT, SIMCONNECT_STATE_OFF);  
                            break;  
                    }  
            }  
        }  
    }  
}
```

```

        printf("\nX AXIS is active");
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_SLIDER, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_XAXIS, SIMCONNECT_STATE_ON);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_YAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_RZAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_HAT, SIMCONNECT_STATE_OFF);
        break;

    case 3:
        printf("\nY AXIS is active");
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_SLIDER, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_XAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_YAXIS, SIMCONNECT_STATE_ON);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_RZAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_HAT, SIMCONNECT_STATE_OFF);
        break;

    case 4:
        printf("\nZ ROTATION is active");
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_SLIDER, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_XAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_YAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_RZAXIS, SIMCONNECT_STATE_ON);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_HAT, SIMCONNECT_STATE_OFF);
        break;

    case 5:
        printf("\nHAT is active");
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_SLIDER, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_XAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_YAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_RZAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_HAT, SIMCONNECT_STATE_ON);
        break;
    }

    default:
        break;
}
break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    quit = 1;
    break;
}

default:
    break;
}
}

void testInput()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Joystick Input", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Set up some private events
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_Z);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_SLIDER);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_XAXIS);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_YAXIS);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_RZAXIS);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_HAT);

        // Add all the private events to a notification group
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_Z);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_SLIDER);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_XAXIS);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_YAXIS);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_RZAXIS);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_0, EVENT_HAT);

        // Set a high priority for the group
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

        // Map input events to the private client events
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_Z, "z", EVENT_Z);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_SLIDER, "joystick:0:slider", EVENT_SLIDER);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_XAXIS, "joystick:0:Xaxis", EVENT_XAXIS);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_YAXIS, "joystick:0:Yaxis", EVENT_YAXIS);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_RZAXIS, "joystick:0:RzAxis", EVENT_RZAXIS);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_HAT, "joystick:0:POV", EVENT_HAT);

        // Turn on the Z key
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_Z, SIMCONNECT_STATE_ON);
        hr = SimConnect_SetInputGroupPriority(hSimConnect, INPUT_Z, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

        // Turn all the joystick events off
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_SLIDER, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_XAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_YAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_RZAXIS, SIMCONNECT_STATE_OFF);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_HAT, SIMCONNECT_STATE_OFF);

        while( 0 == quit )
    }
}

```

```
    {
        SimConnect_CallDispatch(hSimConnect, MyDispatchProcJ, NULL);
        Sleep(1);
    }

    hr = SimConnect_Close(hSimConnect);
}
else
    printf("\nFailed to Connect");
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testInput();
    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

MenuItems.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Menu Items Sample
//
// Description:
//           Add one menu item, after it has been selected four times
//           replace it with another menu item
//-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

int     quit = 0;
HANDLE  hSimConnect = NULL;

enum GROUP_ID {
    GROUP_MENU
};

enum EVENT_ID {
    EVENT_MENU_ONE,
    EVENT_MENU_TWO,
};

static int menuUseCount = 0;

void CALLBACK MyDispatchProcMI(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;

            switch(evt->uEventID)
            {
                case EVENT_MENU_ONE:
                    printf("\nMenu item one selected %d", evt->dwData);

                    ++menuUseCount;

                    // Selected four times, so replace item one with item two
                    if (menuUseCount == 4)
                    {
                        hr = SimConnect_MenuDeleteItem(hSimConnect, EVENT_MENU_ONE);
                        hr = SimConnect_RemoveClientEvent(hSimConnect, GROUP_MENU, EVENT_MENU_ONE);

                        hr = SimConnect_MenuAddItem(hSimConnect, "Menu Item Two", EVENT_MENU_TWO, 54321);
                        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_MENU, EVENT_MENU_TWO);
                    }
                    break;

                case EVENT_MENU_TWO:
                    ++menuUseCount;

                    printf("\nMenu item two selected %d", evt->dwData);

                    if (menuUseCount == 6)
                        quit = 1;
                    break;

                default:
                    printf("\nReceived unknown event: %d", evt->uEventID);
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_QUIT:
        {
            quit = 1;
            break;
        }

        default:
            printf("\nReceived ID: %d", pData->dwID);
            break;
    }
}
```

```
}

void testMenuItems()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Menu Items", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Create some private events
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MENU_ONE);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MENU_TWO);

        // Add one menu item
        hr = SimConnect_MenuAddItem(hSimConnect, "Menu Item One", EVENT_MENU_ONE, 12345);

        // Sign up for the notifications
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_MENU, EVENT_MENU_ONE);

        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_MENU, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcMI, NULL);
            Sleep(1);
        }

        // Clean up before exiting
        if (menuItemCount < 4)
            hr = SimConnect_MenuDeleteItem(hSimConnect, EVENT_MENU_ONE); else
            hr = SimConnect_MenuDeleteItem(hSimConnect, EVENT_MENU_TWO);

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testMenuItems();

    return 0;
}
// End of sample
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

MissionAction.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect Mission Action Sample  
//  
// Description:  
//           Link SimConnect client with a Mission.  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int      quit = 0;  
HANDLE   hSimConnect = NULL;  
  
// Need copies of all the GUIDs used in the mission file  
// See "Mission Action Sample" add-on in the project's source directory.  
  
// {B9566244-7047-4C72-BD57-13AE4FBE4FEC}  
static const GUID g_guidCustomAction1 =  
{ 0xb9566244, 0x7047, 0x4c72, { 0xbd, 0x57, 0x13, 0xae, 0x4f, 0xbe, 0x4f, 0xec } };  
  
// {031152BB-2F7A-471F-BB10-C67BEB5D1D68}  
static const GUID g_guidCustomAction2 =  
{ 0x31152bb, 0x2f7a, 0x471f, { 0xbb, 0x10, 0xc6, 0x7b, 0xeb, 0x5d, 0x1d, 0x68 } };  
  
// {593F90B3-EAA3-4C1B-A863-145CAFE7C3B4}  
static const GUID g_guidMissionAction1 =  
{ 0x593f90b3, 0xea3, 0x4c1b, { 0xa8, 0x63, 0x14, 0x5c, 0xaf, 0xe7, 0xc3, 0xb4 } };  
  
// {E863C0ED-AE38-474B-80C8-4BE46EB14B5A}  
static const GUID g_guidMissionAction2 =  
{ 0xe863c0ed, 0xae38, 0x474b, { 0x80, 0xc8, 0x4b, 0xe4, 0x6e, 0xb1, 0x4b, 0x5a } };  
  
enum EVENT_ID {  
    EVENT_MISSION_ACTION,  
    EVENT_MISSION_COMPLETED  
};  
  
void CALLBACK MyDispatchProcMA(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    HRESULT hr;  
  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT_MISSION_COMPLETED:  
  
                    printf("\nMission completed, result = %d", evt->dwData);  
  
                    // Perform any additional processing to complete the mission  
                    // Perhaps writing out a status file that can be read in by another  
                    // mission.  
  
                    break;  
  
                default:  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_CUSTOM_ACTION:  
        {  
            SIMCONNECT_RECV_CUSTOM_ACTION *pCustomAction = (SIMCONNECT_RECV_CUSTOM_ACTION *)pData;  
  
            if (pCustomAction->guidInstanceId == g_guidCustomAction1)  
            {  
                printf("\nCustom Action 1, payload: %s", pCustomAction->szPayLoad);  
  
                // Custom actions can include calls to actions within the mission xml file, though  
                // if this is done we cannot know if the actions have been completed within this  
                // section of code (the actions may initiate triggers and it may be some time  
                // before the sequence is ended).  
            }  
        }  
    }  
}
```

```

        hr = SimConnect_ExecuteMissionAction(hSimConnect, g_guidMissionAction1);
        hr = SimConnect_ExecuteMissionAction(hSimConnect, g_guidMissionAction2);

    }
    else if (pCustomAction->guidInstanceId == g_guidCustomAction2)
    {
        printf("\nCustom Action 2, payload: %s", pCustomAction->szPayLoad);

        // This action simply notifies the Mission system that the first action
        // is complete
        hr = SimConnect_CompleteCustomMissionAction(hSimConnect, g_guidCustomAction1);

    }
    else
    {
        printf("\nUnknown custom action: %p", pCustomAction->guidInstanceId);
    }

    break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    quit = 1;
    break;
}

default:
    break;
}
}

void testMissionAction()
{
    HRESULT hr;

    HANDLE hEventHandle = ::CreateEvent(NULL, FALSE, FALSE, NULL);

    if(hEventHandle == NULL)
    {
        printf("Error: Event creation failed!");
        return;
    }

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Mission Action", NULL, 0, hEventHandle, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Subscribe to the mission completed event
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_MISSION_COMPLETED, "MissionCompleted");

        // Subscribe to a notification when a custom action executes
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_MISSION_ACTION, "CustomMissionActionExecuted");

        // Check for messages only when a Windows event has been received

        while( 0 == quit && ::WaitForSingleObject(hEventHandle, INFINITE) == WAIT_OBJECT_0)
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcMA, NULL);
            Sleep(1);
        }

        CloseHandle(hEventHandle);

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testMissionAction();

    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

NoCallback.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
//  SimConnect No Callback Sample  
//  
//  Description:  
//      Responds to the user aircraft brakes, without a callback function  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE  hSimConnect = NULL;  
  
static enum GROUP_ID10 {  
    GROUP_10,  
};  
  
static enum EVENT_ID10 {  
    EVENT_BRAKES_10,  
};  
  
void testNoCallback()  
{  
    SIMCONNECT_RECV* pData;  
    DWORD cbData;  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "No Callback", NULL, 0, 0, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES_10, "brakes");  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_10, EVENT_BRAKES_10);  
  
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_10, SIMCONNECT_GROUP_PRIORITY_HIGHEST);  
  
        while( 0 == quit )  
        {  
            hr = SimConnect_GetNextDispatch(hSimConnect, &pData, &cbData);  
  
            if (SUCCEEDED(hr))  
            {  
                switch(pData->dwID)  
                {  
                    case SIMCONNECT_RECV_ID_EVENT:  
                    {  
                        // enter code to handle events received in a SIMCONNECT_RECV_EVENT structure.  
                        SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*) pData;  
  
                        switch(evt->uEventID)  
                        {  
                            case EVENT_BRAKES_10:  
                                printf("\nEvent brakes: %d", evt->dwData);  
                                break;  
  
                            default:  
                                break;  
                        }  
                        break;  
                    }  
                    case SIMCONNECT_RECV_ID_QUIT:  
                        // enter code to handle exiting the application  
                        quit = 1;  
                        break;  
  
                    default:  
                        break;  
                }  
            }  
            hr = SimConnect_Close(hSimConnect);  
        }  
    }  
  
    int __cdecl _tmain(int argc, _TCHAR* argv[]){
```

```
    testNoCallback();  
    return 0;  
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

OpenClose.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
//  SimConnect Open and Close Sample  
//  
//  Description:  
//      Opens and immediately Closes SimConnect  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
HANDLE hSimConnect = NULL;  
  
void testOpenClose()  
{  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Open and Close", NULL, 0, 0, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        hr = SimConnect_Close(hSimConnect);  
  
        printf("\nDisconnected from Prepar3D");  
    } else  
        printf("\nFailed to connect to Prepar3D");  
}  
  
int __cdecl _tmain(int argc, _TCHAR* argv[])  
{  
    testOpenClose();  
  
    return 0;  
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

PauseMonitor.cpp

```
#include <windows.h>
#include "SimConnect.h"
#include <stdio.h>

HANDLE hSimConnect;
static enum EVENTS
{
    EVENT_SIM_PAUSED,
    EVENT_SIM_START,
    EVENT_SIM_STOP
};

static enum PAUSESTATE
{
    UNPAUSED=0,
    PAUSED
};

bool RUNSIMCONNECT;

void CALLBACK Dispatch(SIMCONNECT_RECV* pData, DWORD cbData, void* pContext);

int main (int argc, char* argv)
{
    HRESULT hr;
    RUNSIMCONNECT=true;
    if(SUCCEEDED(SimConnect_Open(&hSimConnect, "PauseMonitor", NULL, 0, 0, 0)))
    {
        printf("Connected to SimConnect!\n");

        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_PAUSED, "Pause");
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_STOP, "SimStop");

        while(RUNSIMCONNECT)
        {
            SimConnect_CallDispatch(hSimConnect, Dispatch, NULL);
            Sleep(1);
        }

        printf("Closing connection to SimConnect\n");
        hr = SimConnect_Close(hSimConnect);

    }
    else
        printf("Failed to connect to SimConnect!\n");
}

void CALLBACK Dispatch(SIMCONNECT_RECV* pData, DWORD cbData, void* pContext)
{
    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT* myEvent = (SIMCONNECT_RECV_EVENT*)pData;
            switch(myEvent->uEventID)
            {
                case EVENT_SIM_START:
                    printf("SimConnect SimStart has occurred\n");
                    break;

                case EVENT_SIM_PAUSED:
                    printf("SimConnect believes the simulation is currently ");
                    switch(myEvent->dwData)
                    {
                        case UNPAUSED:
                            printf("UNPAUSED\n");
                            break;

                        case PAUSED:
                            printf("PAUSED\n");
                            break;
                    }
                    break;

                case EVENT_SIM_STOP:
                    printf("SimConnect SimStop has occurred\n");
                    break;
            }
        }
        break;
    }
}
```

```
    case SIMCONNECT_RECV_ID_QUIT:
    {
        printf("SimConnect Quit Received\n");
        RUNSIMCONNECT=false;
    }
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

requestData.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Data Request Sample
//
// Description:
//           After a flight has loaded, request the lat/lon/alt of the user
//           aircraft
//-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

int     quit = 0;
HANDLE hSimConnect = NULL;

struct Struct1
{
    char    title[256];
    double  kohlsmann;
    double  altitude;
    double  latitude;
    double  longitude;
};

static enum EVENT_ID{
    EVENT_SIM_START,
};

static enum DATA_DEFINE_ID {
    DEFINITION_1,
};

static enum DATA_REQUEST_ID {
    REQUEST_1,
};

void CALLBACK MyDispatchProcRD(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;
            switch(evt->uEventID)
            {
                case EVENT_SIM_START:
                    // Now the sim is running, request information on the user aircraft
                    hr = SimConnect_RequestDataOnSimObjectType(hSimConnect, REQUEST_1, DEFINITION_1, 0, SIMCONNECT_SIMOBJECT_TYPE_USER);
                    break;
                default:
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE:
        {
            SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE *pObjData = (SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE*)pData;
            switch(pObjData->dwRequestId)
            {
                case REQUEST_1:
                {
                    DWORD ObjectID = pObjData->dwObjectID;
                    Struct1 *pS = (Struct1*)&pObjData->dwData;
                    if (SUCCEEDED(StringCbLengthA(&pS->title[0], sizeof(pS->title), NULL))) // security check
                    {
                        printf("\nObjectID=%d title=\"%s\"\nLat=%f Lon=%f Alt=%f Kohlsman=%2f", ObjectID, pS->title, pS->latitude, pS->longitude, pS->altitude, pS->kohlsmann );
                    }
                    break;
                }

                default:
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_QUIT:
        {
            quit = 1;
            break;
        }

        default:
            printf("\nReceived:%d",pData->dwID);
            break;
    }
}
```

```
void testDataRequest()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Request Data", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Set up the data definition, but do not yet do anything with it
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "title", NULL, SIMCONNECT_DATATYPE_STRING256);
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Kohisman setting hg", "inHg");
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Altitude", "feet");
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Latitude", "degrees");
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_1, "Plane Longitude", "degrees");

        // Request an event when the simulation starts
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcRD, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testDataRequest();

    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

ReservedKey.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect Reserved Key Sample  
//  
// Description:  
// Try to reserve keys q, a or z.  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int quit = 0;  
HANDLE hSimConnect = NULL;  
  
enum EVENT_ID {  
    EVENT_RESERVE_REQUEST,  
};  
  
void CALLBACK MyDispatchProcRK(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_RESERVED_KEY:  
        {  
            SIMCONNECT_RECV_RESERVED_KEY *rkey = (SIMCONNECT_RECV_RESERVED_KEY*)pData;  
            printf("\nReserved Key: %s %s", rkey->szChoiceReserved, rkey->szReservedKey);  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_QUIT:  
        {  
            quit = 1;  
            break;  
        }  
  
        default:  
            printf("\nReceived ID: %d", pData->dwID);  
            break;  
    }  
}  
  
  
void testReservedKey()  
{  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Reserved Key", NULL, 0, 0, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        // Create a private event  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_RESERVE_REQUEST);  
  
        // Use the private event to request a reserved key  
        hr = SimConnect_RequestReservedKey(hSimConnect, EVENT_RESERVE_REQUEST, "q", "a", "z");  
  
        while( 0 == quit )  
        {  
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcRK, NULL);  
            Sleep(1);  
        }  
  
        hr = SimConnect_Close(hSimConnect);  
    }  
}  
  
int __cdecl _tmain(int argc, _TCHAR* argv[])  
{  
    testReservedKey();  
    return 0;  
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

ScenerySettings.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect Scenery Complexity and Shadow Flag sample  
//  
// If the key combination U+C is typed, a request is sent to request  
// the current scenery complexity.  
//  
// If the key combination U+F is typed, a request is sent to request  
// the current shadow flags.  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE  hSimConnect = NULL;  
  
  
static enum GROUP_ID {  
    GROUP0,  
};  
  
static enum EVENT_ID {  
    EVENT0,  
    EVENT1,  
};  
  
static enum INPUT_ID {  
    INPUT0,  
};  
  
static enum REQUEST_ID {  
    REQUEST0,  
    REQUEST1,  
};  
  
void CALLBACK MyDispatchProc(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    HRESULT hr;  
  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT0:  
                    printf("\nEVENT0: %d", evt->dwData);  
  
                    // Send a request for the scenery complexity  
                    hr = SimConnect_RequestSceneryComplexity( hSimConnect, REQUEST0 );  
                    break;  
  
                case EVENT1:  
                    printf("\nEVENT1: %d", evt->dwData);  
  
                    // Send a request for the shadow flags  
                    hr = SimConnect_RequestShadowFlags( hSimConnect, REQUEST1 );  
                    break;  
  
                default:  
                    printf("\nSIMCONNECT_RECV_EVENT: 0x%08X 0x%08X 0x%X", evt->uEventID, evt->dwData, cbData);  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_SCENERY_COMPLEXITY:  
        {  
            SIMCONNECT_RECV_SCENERY_COMPLEXITY *pComplexity = (SIMCONNECT_RECV_SCENERY_COMPLEXITY*)pData;  
  
            printf("\nSIMCONNECT_RECV_SCENERY_COMPLEXITY RequestID=%d dwSceneryComplexity=%d\n", pComplexity->dwRequestID, pComplexity->dwSceneryComplexity);  
            break;  
        }  
    }  
}
```

```

    case SIMCONNECT_RECV_ID_SHADOW_FLAGS:
    {
        SIMCONNECT_RECV_SHADOW_FLAGS *pShadows = (SIMCONNECT_RECV_SHADOW_FLAGS*)pData;
        printf("\nSIMCONNECT_RECV_SHADOW_FLAGS RequestID=%d dwShadowFlags=%d\n", pShadows->dwRequestID, pShadows->dwShadowFlags);
        break;
    }

    case SIMCONNECT_RECV_ID_QUIT:
    {
        quit = 1;
        break;
    }

    case SIMCONNECT_RECV_ID_EXCEPTION:
    {
        SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;
        printf("\n***** EXCEPTION=%d SendID=%d uOffset=%d cbData=%d\n", except->dwException, except->dwSendID, except->dwIndex, cbData);
        break;
    }

    default:
        printf("\nUNKNOWN DATA RECEIVED: pData=%p cbData=%d\n", pData, cbData);
        break;
    }
}

//-----  

// main  

//-----  

int __cdecl main(int argc, char* argv[])
{
    HANDLE hEventHandle = ::CreateEvent(NULL, FALSE, FALSE, NULL);
    if(hEventHandle == NULL)
    {
        printf("Error: Event creation failed! Bailing");
        return 1;
    }

    if (FAILED(SimConnect_Open(&hSimConnect, "Scenery Complexity and Shadow Flag Sample", NULL, 0, hEventHandle, 0)))
    {
        printf("\nConnection to Prepar3D failed!");
    }
    else
    {
        printf("\nConnected to Prepar3D!");

        HRESULT hr;

        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT0);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT1);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT0);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT1);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "U+C", EVENT0);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "U+F", EVENT1);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT0, SIMCONNECT_STATE_ON);

        while( 0 == quit && hr==S_OK && ::WaitForSingleObject(hEventHandle, INFINITE) == WAIT_OBJECT_0 )
        {
            hr = SimConnect_CallDispatch(hSimConnect, MyDispatchProc, NULL);
        }
        hr = SimConnect_Close(hSimConnect);
        CloseHandle(hEventHandle);
    }
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

SendEventA.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
// SimConnect Send Event A Sample  
//  
// Description:  
// Whenever the brakes are hit, sends two custom client events to  
// all other clients, one of the events is maskable.  
// Send Event B and C should receive these events.  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int quit = 0;  
HANDLE hSimConnect = NULL;  
  
static enum GROUP_ID {  
    GROUP_A,  
};  
  
static enum EVENT_ID {  
    EVENT_BRAKES,  
    EVENT_MY_EVENT,  
    EVENT_MASKABLE,  
};  
  
void CALLBACK MyDispatchProcA(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    switch(pData->dWID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT_BRAKES:  
                    printf("\nEvent brakes: %d", evt->dwData);  
  
                    // Send the two events to all other client groups - this is achieved by setting the priority of the  
                    // message to SIMCONNECT_GROUP_PRIORITY_HIGHEST. This is the priority of the first client group that  
                    // will be sent the message.  
  
                    SimConnect_TransmitClientEvent(hSimConnect, 0, EVENT_MY_EVENT, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
                    SimConnect_TransmitClientEvent(hSimConnect, 0, EVENT_MASKABLE, 0, SIMCONNECT_GROUP_PRIORITY_HIGHEST, SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY);  
  
                    break;  
  
                case EVENT_MY_EVENT:  
                    printf("\nSend Event A received My.event");  
                    break;  
  
                case EVENT_MASKABLE:  
                    printf("\nSend Event A received My.maskable.event");  
                    break;  
  
                default:  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_QUIT:  
        {  
            quit = 1;  
            break;  
        }  
  
        default:  
            break;  
    }  
}  
  
void testSendEvent()  
{  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Send Event A", NULL, 0, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES, "brakes");  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_A, EVENT_BRAKES);  
  
        // Define two custom events, both of which this client will not mask  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MY_EVENT, "My.event");  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_A, EVENT_MY_EVENT, false);  
  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MASKABLE, "My.maskable.event");  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_A, EVENT_MASKABLE, false);  
  
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_A, SIMCONNECT_GROUP_PRIORITY_HIGHEST);  
  
        while( 0 == quit )  
        {  
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcA, NULL);  
            Sleep(1);  
        }  
  
        hr = SimConnect_Close(hSimConnect);  
    }  
}  
  
int __cdecl _tmain(int argc, _TCHAR* argv[])  
{  
    testSendEvent();  
    return 0;  
}
```

PREPAR3D

SendEventB.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Send Event B Sample
//
// Description:
//     Responds when custom events are sent from the Send Event A sample
//     Masks one of the events from Send Event C
//-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

int quit = 0;
HANDLE hSimConnect = NULL;

static enum GROUP_ID{
    GROUP_B,
};

static enum EVENT_ID {
    EVENT_MY_EVENTB,
    EVENT_MASKABLEB,
};

void CALLBACK MyDispatchProcB(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;

            switch(evt->uEventID)
            {

                case EVENT_MY_EVENTB:
                    printf("\nSend Event B received My.event");
                    break;

                case EVENT_MASKABLEB:
                    printf("\nSend Event B received My.maskable.event");
                    break;

                default:
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_QUIT:
        {
            quit = 1;
            break;
        }

        default:
            break;
    }
}

void testEventReceive()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Send Event B", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Set up to receive the "My.event" notification, without masking it

        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MY_EVENTB, "My.event");
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_B, EVENT_MY_EVENTB, false);

        // Set up to receive the "My.maskable.event" notification, and mask it from lower
        // priority client groups

        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MASKABLEB, "My.maskable.event");
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_B, EVENT_MASKABLEB, true);

        // Set the priority of the group to enable the masking of events
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_B, SIMCONNECT_GROUP_PRIORITY_HIGHEST_MASKABLE);

        while( 0 == quit )
    }
}
```

```
    {
        SimConnect_CallDispatch(hSimConnect, MyDispatchProcB, NULL);
        Sleep(1);
    }

    hr = SimConnect_Close(hSimConnect);
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testEventReceive();
    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

SendEventC.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect Send Event C Sample  
//  
// Description:  
//      Responds when a custom events are sent from the Send Event A sample  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit          = 0;  
HANDLE  hSimConnect   = NULL;  
  
static enum GROUP_ID {  
    GROUP_C,  
};  
  
static enum EVENT_ID {  
    EVENT_MY_EVENTC,  
    EVENT_MASKABLEC,  
};  
  
void CALLBACK MyDispatchProcC(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT_MY_EVENTC:  
                    printf("\nSend Event C received My.event");  
                    break;  
  
                case EVENT_MASKABLEC:  
                    printf("\nSend Event C received My.maskable.event");  
                    break;  
  
                default:  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_QUIT:  
        {  
            quit = 1;  
            break;  
        }  
  
        default:  
            break;  
    }  
}  
  
void testEventReceive()  
{  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Send Event C", NULL, 0, 0, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        // Set up to receive the "My.event" notification, without masking it  
  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MY_EVENTC, "My.event");  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_C, EVENT_MY_EVENTC, false);  
  
        // Set up to receive the "My.maskable.event" notification, and mask it from lower  
        // priority client groups  
  
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_MASKABLEC, "My.maskable.event");  
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_C, EVENT_MASKABLEC, true);  
    }  
}
```

```
// The group priority is set low
hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_C, SIMCONNECT_GROUP_PRIORITY_DEFAULT);

while( 0 == quit )
{
    SimConnect_CallDispatch(hSimConnect, MyDispatchProcC, NULL);
    Sleep(1);
}

hr = SimConnect_Close(hSimConnect);
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testEventReceive();
    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

SetData.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
// SimConnect Set Data Sample  
//  
// Description:  
//           When Ctrl+Shift+A is pressed, the user aircraft is moved  
//           to a new location. When Ctrl+Shift+B is pressed, landing  
//           lights are turned on, health points are increased, and  
//           the main exit is opened halfway.  
//-----  
  
#include <windows.h>  
#include <char.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE hSimConnect = NULL;  
  
enum GROUP_ID {  
    GROUP_INIT,  
    GROUP_STRUCT,  
};  
  
enum INPUT_ID {  
    INPUT_INIT,  
    INPUT_STRUCT,  
};  
  
enum EVENT_ID {  
    EVENT_SIM_START,  
    EVENT_INIT,  
    EVENT_STRUCT,  
};  
  
enum DATA_DEFINE_ID {  
    DEFINITION_INIT,  
    DEFINITION_STRUCT,  
};  
  
// Custom struct used to demonstrate setting different data types.  
struct CustomStruct  
{  
    BOOL    LandingLight;    // SIMCONNECT_DATATYPE_INT32  
    float   Health;        // SIMCONNECT_DATATYPE_FLOAT32  
    double  Exit;          // SIMCONNECT_DATATYPE_FLOAT64  
};  
  
void CALLBACK MyDispatchProcSD(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    HRESULT hr;  
  
    switch (pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch (evt->uEventID)  
            {  
                case EVENT_SIM_START:  
                {  
                    // Turn the input events on now  
                    hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_INIT, SIMCONNECT_STATE_ON);  
                    hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_STRUCT, SIMCONNECT_STATE_ON);  
                }  
                break;  
  
                case EVENT_INIT:  
                {  
                    SIMCONNECT_DATA_INITPOSITION Init;  
                    Init.Altitude = 5000.0;  
                    Init.Latitude = 47.64210;  
                    Init.Longitude = -122.13010;  
                    Init.Pitch = 0.0;  
                    Init.Bank = -1.0;  
                    Init.Heading = 180.0;  
                    Init.OnGround = 0;  
                    Init.Airspeed = 60;  
                    hr = SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION_INIT, SIMCONNECT_OBJECT_ID_USER, 0, 0, sizeof(Init), &Init);  
                    printf("\nEVENT_INIT received and data sent");  
                }  
                break;  
  
                case EVENT_STRUCT:  
                {  
                    CustomStruct Struct;  
                    Struct.LandingLight = 1;  
                    Struct.Health = 1000.0f;  
                    Struct.Exit = 0.5;  
                    hr = SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION_STRUCT, SIMCONNECT_OBJECT_ID_USER, 0, 0, sizeof(Struct), &Struct);  
                    printf("\nEVENT_STRUCT received and data sent");  
                }  
                break;  
  
                default:  
                {  
                    break;  
                }  
                break;  
            }  
        }  
        case SIMCONNECT_RECV_ID_QUIT:  
        {  
            quit = 1;  
            break;  
        }  
    }  
}
```

```

    default:
        printf("\nReceived:%d", pData->dwID);
        break;
    }
}

void testDataSet()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Set Data", NULL, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Set up a data definition for positioning data
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_INIT, "INITIAL POSITION", NULL, SIMCONNECT_DATATYPE_INITPOSITION);

        // Set up data definition for a custom struct with different data types
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_STRUCT, "LIGHT LANDING", NULL, SIMCONNECT_DATATYPE_INT32);
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_STRUCT, "HEALTH POINTS", NULL, SIMCONNECT_DATATYPE_FLOAT32);
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_STRUCT, "EXIT OPEN:0", NULL, SIMCONNECT_DATATYPE_FLOAT64);

        // Request a simulation start event
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

        // Create custom events
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_INIT, "My.CTRLSHIFTA");
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_STRUCT, "My.CTRLSHIFTB");

        // Link the custom event to some keyboard keys, and turn the input event off
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_INIT, "Ctrl+Shift+A", EVENT_INIT);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_INIT, SIMCONNECT_STATE_OFF);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_STRUCT, "Ctrl+Shift+B", EVENT_STRUCT);
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_STRUCT, SIMCONNECT_STATE_OFF);

        // Sign up for notifications for EVENT_INIT and EVENT_STRUCT
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_INIT, EVENT_INIT);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_STRUCT, EVENT_STRUCT);

        while (0 == quit)
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcSD, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testDataSet();
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

SystemEvent.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
///
// Managed System Event sample
//
// Click on the buttons to receive 4 second and Sim start and stop notifications
//

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// Add these two statements to all SimConnect clients
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;

namespace Managed_System_Event
{
    public partial class Form1 : Form
    {

        // User-defined win32 event
        const int WM_USER_SIMCONNECT = 0x0402;

        // SimConnect object
        SimConnect simconnect = null;

        // System state switches
        int P3D_System_4 = 0;
        int P3D_System_Sim = 0;

        enum REQUESTS
        {
            REQUEST_4S,
            REQUEST_SIMSTATE,
        };

        enum EVENTS
        {
            SIMSTART,
            SIMSTOP,
            FOURSECS,
            TEXTWINDOW
        };

        public Form1()
        {
            InitializeComponent();

            setButtons(true, false, false);
        }

        // Simconnect client will send a win32 message when there is
        // a packet to process. ReceiveMessage must be called to
        // trigger the events. This model keeps simconnect processing on the main thread.

        protected override void DefWndProc(ref Message m)
        {
            if (m.Msg == WM_USER_SIMCONNECT)
            {
                if (simconnect != null)
                {
                    simconnect.ReceiveMessage();
                }
            }
            else
            {
                base.DefWndProc(ref m);
            }
        }

        private void setButtons(bool bConnect, bool bGet, bool bDisconnect)
        {
            buttonConnect.Enabled = bConnect;
            buttonRequest4.Enabled = bGet;
            buttonSimStart.Enabled = bGet;
            buttonDisconnect.Enabled = bDisconnect;
        }

        private void closeConnection()
        {
            if (simconnect != null)
```

```

    {
        // Unsubscribe from all the system events
        simconnect.UnsubscribeFromSystemEvent(EVENTS.FOURSECS);
        simconnect.UnsubscribeFromSystemEvent(EVENTS.SIMSTART);
        simconnect.UnsubscribeFromSystemEvent(EVENTS.SIMSTOP);

        // Dispose serves the same purpose as SimConnect_Close()
        simconnect.Dispose();
        simconnect = null;
        displayText("Connection closed");
    }
}

// Set up all the SimConnect related event handlers
private void initSystemEvent()
{
    try
    {
        // listen to connect and quit msgs
        simconnect.OnRecvOpen += new SimConnect.RecvOpenEventHandler(simconnect_OnRecvOpen);
        simconnect.OnRecvQuit += new SimConnect.RecvQuitEventHandler(simconnect_OnRecvQuit);

        // listen to exceptions
        simconnect.OnRecvException += new SimConnect.RecvExceptionEventHandler(simconnect_OnRecvException);

        // listen to events
        simconnect.OnRecvEvent += new SimConnect.RecvEventEventHandler(simconnect_OnRecvEvent);
        simconnect.OnRecvEventText += new SimConnect.RecvEventTextEventHandler(simconnect_OnTextEvent);

        // Subscribe to system events
        simconnect.SubscribeToSystemEvent(EVENTS.FOURSECS, "4sec");
        simconnect.SubscribeToSystemEvent(EVENTS.SIMSTART, "SimStart");
        simconnect.SubscribeToSystemEvent(EVENTS.SIMSTOP, "SimStop");

        // use with TextMenu sample to show an example of listening for text events
        simconnect.SubscribeToSystemEvent(EVENTS.TEXTWINDOW, "TextEventCreated");

        // Initially turn the events off
        simconnect.SetSystemEventState(EVENTS.FOURSECS, SIMCONNECT_STATE.OFF);
        simconnect.SetSystemEventState(EVENTS.SIMSTART, SIMCONNECT_STATE.OFF);
        simconnect.SetSystemEventState(EVENTS.SIMSTOP, SIMCONNECT_STATE.OFF);
    }
    catch (COMException ex)
    {
        displayText(ex.Message);
    }
}

void simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
{
    displayText("Connected to Prepar3D");
}

// The case where the user closes Prepar3D
void simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
{
    displayText("Prepar3D has exited");
    closeConnection();
}

// The case where the user closes the client
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    buttonDisconnect_Click(sender, null);
}

void simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    displayText("Exception received: " + data.dwException);
}

void simconnect_OnRecvEvent(SimConnect sender, SIMCONNECT_RECV_EVENT recEvent)
{
    switch (recEvent.uEventID)
    {
        case (uint)EVENTS.SIMSTART:

            displayText("Sim running");
            break;

        case (uint)EVENTS.SIMSTOP:

            displayText("Sim stopped");
            break;

        case (uint)EVENTS.FOURSECS:

            displayText("4s tick");
            break;
    }
}

void simconnect_OnTextEvent(SimConnect sender, SIMCONNECT_RECV_EVENT_TEXT data)
{
    System.String s = "";
    for (uint i = 0; i < data.dwUnitSize; ++i)

```

```

        {
            System.Byte? b = (data.rgMessage[i] as System.Byte?);
            if (b != null)
            {
                System.Char? c = Convert.ToChar(b);
                if (c != null)
                {
                    if (c == '\0')
                    {
                        s += '\n';
                    }
                    else
                    {
                        s += c;
                    }
                }
            }
        }

        if (!String.IsNullOrWhiteSpace(s))
        {
            displayText(s);
        }
    }

    private void buttonConnect_Click(object sender, EventArgs e)
    {
        if (simconnect == null)
        {
            try
            {
                // the constructor is similar to SimConnect_Open in the native API
                simconnect = new SimConnect("Managed System Event", this.Handle, WM_USER_SIMCONNECT, null, 0);

                setButtons(false, true, true);

                initSystemEvent();

            }
            catch (COMException ex)
            {
                displayText("Unable to connect to Prepar3D " + ex.Message);
            }
        }
        else
        {
            displayText("Error - try again");
            closeConnection();

            setButtons(true, false, false);
        }
    }

    private void buttonDisconnect_Click(object sender, EventArgs e)
    {
        // If they are on, turn off the system event subscriptions
        if (P3D_System_4 == 1)
            buttonRequest4_Click(sender, null);
        if (P3D_System_Sim == 1)
            buttonSimStart_Click(sender, null);

        closeConnection();
        setButtons(true, false, false);
    }

    private void buttonRequest4_Click(object sender, EventArgs e)
    {
        // Toggle switch
        P3D_System_4 = 1 - P3D_System_4;

        if (P3D_System_4 == 1)
        {
            simconnect.SetSystemEventState(EVENTS.FOURSECS, SIMCONNECT_STATE.ON);

            buttonRequest4.Text = "Stop 4 sec event";
        }
        else
        {
            simconnect.SetSystemEventState(EVENTS.FOURSECS, SIMCONNECT_STATE.OFF);

            buttonRequest4.Text = "Request 4 sec event";
        }
    }

    private void buttonSimStart_Click(object sender, EventArgs e)
    {
        // Toggle switch
        P3D_System_Sim = 1 - P3D_System_Sim;

        if (P3D_System_Sim == 1)
        {
            simconnect.SetSystemEventState(EVENTS.SIMSTART, SIMCONNECT_STATE.ON);
            simconnect.SetSystemEventState(EVENTS.SIMSTOP, SIMCONNECT_STATE.ON);

            buttonSimStart.Text = "Stop sim events";
        }
        else
        {
            simconnect.SetSystemEventState(EVENTS.SIMSTART, SIMCONNECT_STATE.OFF);
        }
    }
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

TaggedData.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect Tagged Data Request Sample  
//  
// Description:  
//           After a flight has loaded, request the vertical speed and pitot  
//           heat switch setting of the user aircraft, but only when the data  
//           has changed  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE  hSimConnect = NULL;  
  
// A basic structure for a single item of returned data  
struct StructOneDatum {  
    int      id;  
    float    value;  
};  
  
// maxReturnedItems is 2 in this case, as the sample only requests  
// vertical speed and pitot heat switch data  
#define maxReturnedItems    2  
  
// A structure that can be used to receive Tagged data  
struct StructDatum {  
    StructOneDatum datum[maxReturnedItems];  
};  
  
static enum EVENT_PDR {  
    EVENT_SIM_START,  
};  
  
static enum DATA_DEFINE_ID {  
    DEFINITION_PDR,  
};  
  
static enum DATA_REQUEST_ID {  
    REQUEST_PDR,  
};  
  
static enum DATA_NAMES {  
    DATA_VERTICAL_SPEED,  
    DATA_PITOT_HEAT,  
};  
  
void CALLBACK MyDispatchProcPDR(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    HRESULT hr;  
  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
            switch(evt->uEventID)  
            {  
                case EVENT_SIM_START:  
  
                    // Make the call for data every second, but only when it changes and  
                    // only that data that has changed  
                    hr = SimConnect_RequestDataOnSimObject(hSimConnect, REQUEST_PDR, DEFINITION_PDR,  
                        SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD_SECOND,  
                        SIMCONNECT_DATA_REQUEST_FLAG_CHANGED | SIMCONNECT_DATA_REQUEST_FLAG_TAGGED );  
  
                    break;  
  
                default:  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:  
        {  
            SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData = (SIMCONNECT_RECV_SIMOBJECT_DATA*)pData;
```

```

        switch(pObjData->dwRequestID)
    {
        case REQUEST_PDR:
        {
            int count = 0;
            StructDatum *pS = (StructDatum*)&pObjData->dwData;

            // There can be a minimum of 1 and a maximum of maxReturnedItems
            // in the StructDatum structure. The actual number returned will
            // be held in the dwDefineCount parameter.

            while (count < (int) pObjData->dwDefineCount)
            {
                switch (pS->datum[count].id)
                {
                    case DATA_VERTICAL_SPEED:
                        printf("\nVertical speed = %f", pS->datum[count].value );
                        break;

                    case DATA_PITOT_HEAT:
                        printf("\nPitot heat = %f", pS->datum[count].value );
                        break;

                    default:
                        printf("\nUnknown datum ID: %d", pS->datum[count].id);
                        break;
                }
                ++count;
            }
            break;
        }

        default:
            break;
    }
    break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    quit = 1;
    break;
}

default:
    printf("\nUnknown dwID: %d",pData->dwID);
    break;
}
}

void testTaggedDataRequest()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Tagged Data", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Set up the data definition, ensuring that all the elements are in Float32 units, to
        // match the StructDatum structure
        // The number of entries in the DEFINITION_PDR definition should be equal to
        // the maxReturnedItems define

        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_PDR, "Vertical Speed", "Feet per second",
                                            SIMCONNECT_DATATYPE_FLOAT32, 0, DATA_VERTICAL_SPEED);
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_PDR, "Pitot Heat", "Bool",
                                            SIMCONNECT_DATATYPE_FLOAT32, 0, DATA_PITOT_HEAT);

        // Request a simulation start event
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcPDR, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testTaggedDataRequest();

    return 0;
}

```

herein may be the trademarks or their respective owners.

PREPAR3D

```
textmenu.cpp
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Text Menu sample
//
// Description:
//           Ctrl F1 displays a menu on the screen
//           Ctrl F2 removes the menu from the screen
//           Selecting any menu option sends an event and removes the menu
//-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>
#include <string>

enum GROUP_ID {
    GROUP0,
};

enum EVENT_ID {
    EVENT1,
    EVENT2,
    EVENT_MENU_1,
    EVENT_TEXT_1,
    EVENT_MESSAGE_1,
    EVENT_TEXT_WINDOW_CREATED,
};

enum INPUT_ID {
    INPUT0,
};

int quit = 0;
HANDLE hSimConnect = NULL;
static const char Empty1[] = "";
static const char Menu1[] = "SimConnect Text Menu\0Choose which item:\0Item #1\0Item #2\0Item #3\0Item #4\0Item #5\0";
static const char Text1[] = "Sample scrolling text.";
static const char Message1[] = "This is a sample message window.\n\nMessage windows can display more than one line.";

const char* MenuText(SIMCONNECT_TEXT_RESULT result)
{
    switch (result)
    {
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_1:
            return "Item #1 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_2:
            return "Item #2 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_3:
            return "Item #3 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_4:
            return "Item #4 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_5:
            return "Item #5 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_6:
            return "Item #6 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_7:
            return "Item #7 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_8:
            return "Item #8 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_9:
            return "Item #9 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_SELECT_10:
            return "Item #10 Selected";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_DISPLAYED:
            return "Displayed";
            break;
        case SIMCONNECT_TEXT_RESULT_MENU_QUEUED:
            return "Queued";
            break;
        case SIMCONNECT_TEXT_RESULT_REMOVED:
            return "Removed from Queue";
            break;
        case SIMCONNECT_TEXT_RESULT_REPLACE:
            return "Replaced in Queue";
            break;
        case SIMCONNECT_TEXT_RESULT_TIMEOUT:
            return "Timeout";
            break;
        default:
            return "<unknown>";
            break;
    }
}

void CALLBACK MySignalProc(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    switch (pData->dwID)
    {
        case SIMCONNECT_RECV_ID_OPEN:
        {
            SIMCONNECT_RECV_OPEN *pOpen = (SIMCONNECT_RECV_OPEN*)pData;
            printf("Open: AppName=%"s" AppVersion=%d.%d.%d.%d SimConnectVersion=%d.%d.%d.%d\n", pOpen->szApplicationName,
```

```

>pOpen->dwApplicationVersionMajor, pOpen->dwApplicationVersionMinor, pOpen->dwApplicationBuildMajor, pOpen-
>dwApplicationBuildMinor,
    pOpen->dwSimConnectVersionMajor, pOpen->dwSimConnectVersionMinor, pOpen->dwSimConnectBuildMajor, pOpen-
>dwSimConnectBuildMinor
);
break;
}

case SIMCONNECT_RECV_ID_EVENT:
{
    SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;

    switch (evt->uEventID)
    {
        case EVENT1:
            // Display menu
            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_1, sizeof(Menu1), (void*)Menu1);
            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_SCROLL_BLUE, 0, EVENT_TEXT_1, sizeof(Text1), (void*)Text1);
            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW, 0, EVENT_MESSAGE_1, sizeof(Message1), (void*)Message1);
            break;

        case EVENT2:
            // Stop displaying menu
            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MENU, 0, EVENT_MENU_1, sizeof(Empty1), (void*)Empty1);
            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_SCROLL_BLUE, 0, EVENT_TEXT_1, sizeof(Empty1), (void*)Empty1);
            SimConnect_Text(hSimConnect, SIMCONNECT_TEXT_TYPE_MESSAGE_WINDOW, 0, EVENT_MESSAGE_1, sizeof(Empty1), (void*)Empty1);
            break;

        case EVENT_MENU_1:
            // Respond to the users menu selection
            printf("\n");
            printf(MenuText((SIMCONNECT_TEXT_RESULT)evt->dwData));
            break;

        default:
            printf("\nSIMCONNECT_RECV_EVENT: 0x%08X 0x%08X 0x%X", evt->uEventID, evt->dwData, cbData);
            break;
    }
    break;
}

case SIMCONNECT_RECV_ID_EVENT_TEXT:
{
    SIMCONNECT_RECV_EVENT_TEXT* textData = (SIMCONNECT_RECV_EVENT_TEXT*)pData;

    if (textData->eTextType == SIMCONNECT_TEXT_TYPE_MENU)
    {
        void* message = &textData->rgMessage;
        int remainingSize = textData->dwUnitSize;

        for (int itemPosition = 0; remainingSize > 0; ++itemPosition)
        {
            int itemLength = strlen((char*)message, remainingSize) + sizeof(char);

            if (itemLength <= sizeof(char))
            {
                break;
            }

            remainingSize -= itemLength;

            std::string messageItem = reinterpret_cast<char*> (message);

            switch (itemPosition)
            {
                case 0:
                    printf("\nSIMCONNECT_RECV_ID_EVENT_TEXT: SIMCONNECT_TEXT_TYPE_MENU\nTitle: \"%s\"\n", messageItem.c_str());
                    break;
                case 1:
                    printf("Prompt: \"%s\"\n", messageItem.c_str());
                    break;
                default:
                    printf("Option: \"%s\"\n", messageItem.c_str());
                    break;
            }

            message = (BYTE*)((int)message + itemLength);
        }
    }
    else
    {
        std::string message = reinterpret_cast<char*> (textData->rgMessage);
        printf("\nSIMCONNECT_RECV_ID_EVENT_TEXT: \n \"%s\"\n", message.c_str());
        break;
    }
    break;
}

case SIMCONNECT_RECV_ID_QUIT:
{
    printf("\n***** SIMCONNECT_RECV_ID_QUIT *****\n");
    quit = 1;
    break;
}

case SIMCONNECT_RECV_ID_EXCEPTION:
{
    SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;
    printf("\n\n***** EXCEPTION=%d SendID=%d uOffset=%d cbData=%d\n", except->dwException, except->dwSendID, except-
>dwIndex, cbData);
    break;
}

default:
    printf("\nUNKNOWN DATA RECEIVED: pData=%p cbData=%d\n", pData, cbData);
    break;
}
}

//-----
// main
//-----
int __cdecl main(int argc, char* argv[])
{
    HANDLE hEventHandle = ::CreateEvent(NULL, FALSE, FALSE, NULL);

    if (hEventHandle == NULL)
    {
        printf("Error: Event creation failed! Quitting");
        return GetLastError();
    }
}

```

```
}

if (FAILED(SimConnect_Open(&hSimConnect, "SimConnect_Text", NULL, 0, hEventHandle, 0)))
{
    printf("\nConnection to Prepar3D failed! Quitting");
    return 1;
}

printf("\nConnected to Prepar3D");

HRESULT hr;

hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT1);
hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT2);

hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT1, TRUE);
hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP0, EVENT2, TRUE);

hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP0, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "Ctrl+F1", EVENT1);
hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT0, "Ctrl+F2", EVENT2);

hr = SimConnect_SetInputGroupState(hSimConnect, INPUT0, SIMCONNECT_STATE_ON);

hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_TEXT_WINDOW_CREATED, "TextEventCreated");

while (0 == quit && ::WaitForSingleObject(hEventHandle, INFINITE) == WAIT_OBJECT_0)
{
    hr = SimConnect_CallDispatch(hSimConnect, MySignalProc, NULL);
    if (FAILED(hr))
    {
        break;
    }
}

hr = SimConnect_Close(hSimConnect);
CloseHandle(hEventHandle);

return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

```
ThrottleControl.cpp
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Throttle Control sample
// Description:
//           Press 1 to increase the throttle
//           Press 2 to decrease the throttle
//-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include "SimConnect.h"
#include <strsafe.h>

int quit = 0;
HANDLE hSimConnect = NULL;

enum GROUP_ID{
    GROUP_KEYS,
};

enum INPUT_ID {
    INPUT_KEYS,
};

enum EVENT_ID {
    EVENT_SIM_START,
    EVENT_1,
    EVENT_2
};

enum DATA_DEFINE_ID {
    DEFINITION_THROTTLE,
};

enum DATA_REQUEST_ID {
    REQUEST_THROTTLE,
};

struct structThrottleControl
{
    double throttlePercent;
};

structThrottleControl tc;

void CALLBACK MyDispatchProcTC(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:
        {
            SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData = (SIMCONNECT_RECV_SIMOBJECT_DATA*)pData;
            switch(pObjData->dwRequestID)
            {
                case REQUEST_THROTTLE:
                {
                    // Read and set the initial throttle control value
                    structThrottleControl *pS = (structThrottleControl*)&pObjData->dwData;
                    tc.throttlePercent = pS->throttlePercent;

                    printf("\nREQUEST_USERID received, throttle = %2.1f", pS->throttlePercent);

                    // Now turn the input events on
                    hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_KEYS, SIMCONNECT_STATE_ON);
                }
                default:
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;
            switch(evt->uEventID)
            {
                case EVENT_SIM_START:
                {
                    // Send this request to get the user aircraft id
                    hr = SimConnect_RequestDataOnSimObject(hSimConnect, REQUEST_THROTTLE, DEFINITION_THROTTLE, SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD_ONCE);
                }
                break;

                case EVENT_1:
                {
                    // Increase the throttle
                    if (tc.throttlePercent <= 95.0f)
                        tc.throttlePercent += 5.0f;

                    hr = SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION_THROTTLE, SIMCONNECT_OBJECT_ID_USER, 0, 0, sizeof(tc), &tc);
                }
                break;

                case EVENT_2:
                {
                    // Decrease the throttle
                    if (tc.throttlePercent >= 5.0f)
                        tc.throttlePercent -= 5.0f;

                    hr = SimConnect_SetDataOnSimObject(hSimConnect, DEFINITION_THROTTLE, SIMCONNECT_OBJECT_ID_USER, 0, 0, sizeof(tc), &tc);
                }
                break;
            }
            default:
                break;
        }
        case SIMCONNECT_RECV_ID_QUIT:
        {
            quit = 1;
            break;
        }
    default:
        printf("\nReceived:%d",pData->dwID);
        break;
    }
}
```

```

    }

void testThrottleControl()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Throttle Control", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Set up a data definition for the throttle control
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_THROTTLE,
            "GENERAL ENG THROTTLE LEVER POSITION:1", "percent");

        // Request a simulation started event
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

        // Create two private key events to control the throttle
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_1);
        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_2);

        // Link the events to some keyboard keys
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_KEYS, "1", EVENT_1);
        hr = SimConnect_MapInputEventToClientEvent(hSimConnect, INPUT_KEYS, "2", EVENT_2);

        // Ensure the input events are off until the sim is up and running
        hr = SimConnect_SetInputGroupState(hSimConnect, INPUT_KEYS, SIMCONNECT_STATE_OFF);

        // Sign up for notifications
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_KEYS, EVENT_1);
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_KEYS, EVENT_2);

        // Set a high priority for the group
        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_KEYS, SIMCONNECT_GROUP_PRIORITY_HIGHEST);

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcTC, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testThrottleControl();

    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
 The names of the actual companies and products mentioned
 herein may be the trademarks of their respective owners.

PREPAR3D

TrackingErrors.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//  
// SimConnect Tracking Errors Sample  
//  
// Description:  
//           Shows how to use GetLastSendID to record the ID of a request, along  
//           with an identification string, in order to match the IDs of errors  
//           returned to identify which call caused the error  
//  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int     quit = 0;  
HANDLE  hSimConnect = NULL;  
  
#define max_send_records      10  
  
// Declare a structure to hold the send IDs and identification strings  
struct  record_struct {  
    char   call[256];  
    DWORD  sendid;  
};  
  
int     record_count = 0;  
struct  record_struct send_record[max_send_records];  
  
// Record the ID along with the identification string in the send_record structure  
void addSendRecord(char* c)  
{  
    DWORD id;  
  
    if (record_count < max_send_records)  
    {  
        int hr = SimConnect_GetLastSentPacketID(hSimConnect, &id);  
  
        strncpy_s(send_record[ record_count ].call, 255, c, 255);  
        send_record[ record_count ].sendid = id;  
        ++record_count;  
    }  
}  
  
// Given the ID of an erroneous packet, find the identification string of the call  
char* findSendRecord(DWORD id)  
{  
    bool found  = false;  
    int count  = 0;  
    while (!found && count < record_count)  
    {  
        if (id == send_record[count].sendid)  
            return send_record[count].call;  
        ++count;  
    }  
    return "Send Record not found";  
}  
  
static enum GROUP_ID11 {  
    GROUP_11,  
};  
  
static enum EVENT_ID11 {  
    EVENT_BRAKES_11,  
    EVENT_BAD,  
};  
  
void CALLBACK MyDispatchProc11(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
  
            switch(evt->uEventID)  
            {  
                case EVENT_BRAKES_11:  
                    printf("\nEvent brakes: %d", evt->dwData);  
                    break;  
  
                default:  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_EXCEPTION:  
        {  
            SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;
```

```

        printf("\n\n***** EXCEPTION=%d  SendID=%d  Index=%d  cbData=%d\n", except->dwException, except->dwSendID, except-
>dwIndex, cbData);

        // Locate the bad call and print it out
        char* s = findSendRecord(except->dwSendID);
        printf("\n%s", s);
        break;
    }

    case SIMCONNECT_RECV_ID_QUIT:
    {
        quit = 1;
        break;
    }

    default:
        break;
}
}

void testErrors()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Tracking Errors", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        hr = SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES_11, "brakes");
        addSendRecord(" SimConnect_MapClientEventToSimEvent(hSimConnect, EVENT_BRAKES_11, \"brakes\"); ");

        // To force an error, use the wrong event
        hr = SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_11, EVENT_BAD);
        addSendRecord(" SimConnect_AddClientEventToNotificationGroup(hSimConnect, GROUP_11, EVENT_BAD); ");

        hr = SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_11, SIMCONNECT_GROUP_PRIORITY_HIGHEST);
        addSendRecord(" SimConnect_SetNotificationGroupPriority(hSimConnect, GROUP_11, SIMCONNECT_GROUP_PRIORITY_HIGHEST); ");

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProc11, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testErrors();
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

```
VariableStrings.cpp
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//-----
// SimConnect Variable String Sample
// Description:
// Shows how to extract three variable length strings from a
// structure
//-----

#include <windows.h>
#include <tchar.h>
#include <stdio.h>

#include "SimConnect.h"

static enum EVENT_ID {
    EVENT_SIM_START,
};

static enum DATA_DEFINE_ID {
    DEFINITION_1
};

static enum DATA_REQUEST_ID {
    REQUEST_1
};

struct StructVS
{
    BYTE     strings[1]; // variable-length strings
};

int     quit          = 0;
HANDLE hSimConnect   = NULL;

void CALLBACK MyDispatchProcVS(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;
            switch(evt->uEventID)
            {
                case EVENT_SIM_START:

                    // Send this request to get the user aircraft id
                    HRESULT hr = SimConnect_RequestDataOnSimObjectType(hSimConnect, REQUEST_1, DEFINITION_1, 0, SIMCONNECT_SIMOBJECT_TYPE_USER);
                    break;
            }
            break;
        }

        case SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE:
        {
            SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE *pObjData = (SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE*)pData;

            switch(pObjData->dwRequestID)
            {
                case REQUEST_1:
                {
                    StructVS *pS = (StructVS*)&pObjData->dwData;
                    wchar_t*   pszTitle; // wide string
                    char*      pszAirline;
                    char*      pszType;
                    DWORD      cbTitle;
                    DWORD      cbAirline;
                    DWORD      cbType;

                    // Note how the third parameter is moved along the data received.
                    // Reinterpret cast used to ensure pointer arithmetic is correct when using wide strings.
                    if (SUCCEEDED(SimConnect_RetrieveStringW(pData, cbData, &pS->strings, &pszTitle, &cbTitle)) &&
                        SUCCEEDED(SimConnect_RetrieveString(pData, cbData, reinterpret_cast<BYTE*>(pszTitle) + cbTitle, &pszAirline, &cbAirline)) &&
                        SUCCEEDED(SimConnect_RetrieveString(pData, cbData, reinterpret_cast<BYTE*>(pszAirline) + cbAirline, &pszType, &cbType)))
                    {
                        // Note: Using %S string formatter rather than %s for the wide string title.
                        printf("\nTitle = \"%S\" \nAirline = \"%s\" \nType = \"%s\"", pszTitle, pszAirline, pszType);
                    }
                    else
                    {
                        printf("\nCouldn't retrieve the strings.");
                    }
                    break;
                }
                break;
            }
        }

        case SIMCONNECT_RECV_ID_QUIT:
        {
            quit = 1;
            break;
        }

        case SIMCONNECT_RECV_ID_EXCEPTION:
        {
            SIMCONNECT_RECV_EXCEPTION *except = (SIMCONNECT_RECV_EXCEPTION*)pData;
            printf("\n***** EXCEPTION=%d  SendID=%d  Index=%d  cbData=%d\n", except->dwException, except->dwSendID, except->dwIndex, cbData);
            break;
        }

        default:
            printf("\nUNKNOWN DATA RECEIVED: pData=%p cbData=%d\n", pData, cbData);
            break;
    }
}

bool testVariableStrings()
{
    HANDLE hEventHandle = ::CreateEvent(NULL, FALSE, FALSE, NULL);
```

```
if(hEventHandle == NULL)
{
    printf("Error: Event creation failed!");
    return false;
}

HRESULT hr;

if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Variable Strings", NULL, 0, hEventHandle, 0)))
{
    printf("\nConnected to Prepar3D!");

    // Set up a data definition contained a number of variable length strings
    hr = SimConnect_AddDataDefinition(hSimConnect, DEFINITION_1, "TITLE",
        NULL, SIMCONNECT_DATATYPE_WSTRINGV); // wide string
    hr = SimConnect_AddDataDefinition(hSimConnect, DEFINITION_1, "ATC AIRLINE",
        NULL, SIMCONNECT_DATATYPE_STRINGV);
    hr = SimConnect_AddDataDefinition(hSimConnect, DEFINITION_1, "ATC TYPE",
        NULL, SIMCONNECT_DATATYPE_STRINGV);

    // Request a simulation start event
    hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

    while( 0 == quit && ::WaitForSingleObject(hEventHandle, INFINITE) == WAIT_OBJECT_0)
    {
        SimConnect_CallDispatch(hSimConnect, MyDispatchProcVS, NULL);
    }

    CloseHandle(hEventHandle);
    hr = SimConnect_Close(hSimConnect);
    return true;
}
return false;
}

int __cdecl _tmain(int argc, char* argv[])
{
    bool ok = testVariableStrings();
    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

WeatherStation.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//  
// SimConnect Weather Station sample  
//  
// Description:  
// Requests weather data from the nearest weather station to the  
// user aircraft, every 10 seconds  
//  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include "SimConnect.h"  
#include <strsafe.h>  
  
int quit = 0;  
HANDLE hSimConnect = NULL;  
  
static enum EVENT_ID {  
    EVENT_SIM_START,  
};  
  
static enum DATA_DEFINE_ID7 {  
    DEFINTION_LLA,  
};  
  
static enum DATA_REQUEST_ID7 {  
    REQUEST_LLA,  
    REQUEST_WEATHER,  
};  
  
struct Struct_7  
{  
    double altitude;  
    double latitude;  
    double longitude;  
};  
  
void CALLBACK MyDispatchProc7(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    HRESULT hr;  
  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
            switch(evt->uEventID)  
            {  
                case EVENT_SIM_START:  
  
                    // Start requesting lat/lon data every 10 seconds  
                    hr = SimConnect_RequestDataOnSimObject(hSimConnect, REQUEST_LLA, DEFINTION_LLA, SIMCONNECT_OBJECT_ID_USER,  
                                                SIMCONNECT_PERIOD_SECOND, 0, 0, 10, 0);  
                    break;  
                }  
                break;  
            }  
  
        case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:  
        {  
            SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData = (SIMCONNECT_RECV_SIMOBJECT_DATA*)pData;  
  
            printf("\nObject data received");  
  
            switch(pObjData->dwRequestId)  
            {  
                case REQUEST_LLA:  
                {  
                    Struct_7 *pS = (Struct_7*)&pObjData->dwData;  
                    printf("\nLat=%f Lon=%f IndAlt=%f", pS->latitude, pS->longitude, pS->altitude );  
  
                    // Now request the weather data - this will also be requested every 10 seconds  
  
                    hr = SimConnect_WeatherRequestObservationAtNearestStation(hSimConnect, REQUEST_WEATHER, (float) pS->latitude, (float) pS->longitude);  
  
                    break;  
                }  
  
                default:  
                break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_WEATHER_OBSERVATION:  
        {  
            SIMCONNECT_RECV_WEATHER_OBSERVATION* pWxData = (SIMCONNECT_RECV_WEATHER_OBSERVATION*) pData;  
  
            const char* pszMETAR = pWxData->szMetar;
```

```

        switch(pWxData->dwRequestID)
        {
            case REQUEST_WEATHER:
                printf("\n\nWEATHER OBSERVATION: %s", pszMETAR);
                break;

            }
            break;
        }

        case SIMCONNECT_RECV_ID_QUIT:
        {
            quit = 1;
            break;
        }

        default:
            printf("\nReceived:%d",pData->dwID);
            break;
    }
}

void testWeatherNearestStation()
{
    HRESULT hr;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Weather Station", NULL, 0, 0, 0)))
    {
        printf("\nConnected to Prepar3D!");

        // Set up the data definition, note this matches the order in Struct_7
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_LLA, "Indicated Altitude", "feet");
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_LLA, "Plane Latitude", "degrees");
        hr = SimConnect_AddToDataDefinition(hSimConnect, DEFINITION_LLA, "Plane Longitude", "degrees");

        // Request a flight loaded event
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart");

        while( 0 == quit )
        {
            SimConnect_CallDispatch(hSimConnect, MyDispatchProc7, NULL);
            Sleep(1);
        }

        hr = SimConnect_Close(hSimConnect);
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    testWeatherNearestStation();
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

WindowsEvent.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
//  
// SimConnect Windows Event Sample  
//  
// Description:  
// Requests a four second timing event, and implements a Windows  
// Event handler to minimize processing time  
//-----  
  
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
  
#include "SimConnect.h"  
  
static enum EVENT_ID {  
    EVENT_4S,  
};  
  
int quit = 0;  
HANDLE hSimConnect = NULL;  
  
static int tick = 0;  
  
void CALLBACK MyDispatchProcWE(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)  
{  
    switch(pData->dwID)  
    {  
        case SIMCONNECT_RECV_ID_EVENT:  
        {  
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;  
            switch(evt->uEventID)  
            {  
                case EVENT_4S:  
                    printf("\n4 second timer: %d", ++tick);  
                    break;  
  
                default:  
                    printf("\nSIMCONNECT_RECV_EVENT: 0x%08X 0x%08X 0x%X", evt->uEventID, evt->dwData, cbData);  
                    break;  
            }  
            break;  
        }  
  
        case SIMCONNECT_RECV_ID_QUIT:  
        {  
            quit = 1;  
            break;  
        }  
  
        default:  
            printf("\nUNKNOWN DATA RECEIVED: pData=%p cbData=%d\n", pData, cbData);  
            break;  
    }  
}  
  
bool testWindowsEvent()  
{  
    HANDLE hEventHandle = ::CreateEvent(NULL, FALSE, FALSE, NULL);  
    if(hEventHandle == NULL)  
    {  
        printf("Error: Event creation failed!");  
        return false;  
    }  
  
    HRESULT hr;  
  
    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "Windows Event", NULL, 0, hEventHandle, 0)))  
    {  
        printf("\nConnected to Prepar3D!");  
  
        // Subscribe to the four second timer  
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_4S, "4sec");  
  
        // Check for messages only when a Windows event has been received  
        while( 0 == quit && ::WaitForSingleObject(hEventHandle, INFINITE) == WAIT_OBJECT_0)  
        {  
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcWE, NULL);  
        }  
    }  
}
```

```
        }

        CloseHandle(hEventHandle);
        hr = SimConnect_Close(hSimConnect);
        return true;
    }
    return false;
}

int __cdecl _tmain(int argc, char* argv[])
{
    bool ok = testWindowsEvent();
    return 0;
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

```
DestroyerConnector.cs
using System;
using System.Collections.Generic;
using System.Threading;

using LockheedMartin.Prepar3D.SimConnect;

namespace AIDestroyer
{
    class DestroyerConnector : SimConnector
    {
        #region Constants
        /// <summary>
        /// The maximum radius allowed for querying AI traffic as stated in the SDK.
        /// </summary>
        private const uint MAXIMUM_RADIUS = 200000;
        #endregion

        #region Events
        /// <summary>
        /// Fired off when a simobject request is received.
        /// </summary>
        public event EventHandler<EventArgs<SimObject>> SimObjectDataRequests;

        /// <summary>
        /// Fired off when a health status request is received.
        /// </summary>
        public event EventHandler<EventArgs<HealthUpdate>> HealthStatusRequests;
        #endregion

        #region Fields
        /// <summary>
        /// The current aircraft. This is used in setting values.
        /// Mostly just need the ObjectID from it.
        /// </summary>
        private SimObject current_object;
        #endregion

        #region Constructors
        /// <summary>
        /// Create a new simulation connection.
        /// </summary>
        /// <param name="connectionName">The name for this connection.</param>
        public DestroyerConnector(string connectionName) : base(connectionName)
        {
        }
        #endregion

        #region Methods
        /// <summary>
        /// Get the SimObject data for all the simulation objects near the user's aircraft.
        /// </summary>
        public void GetSimObjectData()
        {
            // Only do this if there is a valid simulation connection.
            if (simulation_connection != null)
            {
                try
                {
                    // Get the sim objects at the maximum distance radius allowed.
                    simulation_connection.RequestDataOnSimObjectType(DataIdentifier.RetrieveSimObjects, StructureType.SimObject, MAXIMUM_RADIUS, SIMCONNECT_SIMOBJECT_TYPE.AIRCRAFT);
                }
                catch (Exception ex)
                {
                    SendError(ex.Message);
                }
            }
        }

        public void SetCurrentObject(SimObject simObject)
        {
            UnloadCurrentObjectData();
            current_object = simObject;
        }

        public void SetHealth(long health)
        {
            if (current_object != null)
            {
                // Set the object's health value.
                simulation_connection.SetDataOnSimObject(StructureType.HealthStatus,
                    current_object.ObjectID,
                    SIMCONNECT_DATA_SET_FLAG.DEFAULT,
                    new HealthStatus(health));

                // Now ask for an update on the objects health value.
                simulation_connection.RequestDataOnSimObjectType(DataIdentifier.RetrieveHealthInfo, StructureType.HealthStatus, MAXIMUM_RADIUS, SIMCONNECT_SIMOBJECT_TYPE.AIRCRAFT);
            }
        }

        /// <summary>
        /// Called when the connector connects to Prepar3D.
        /// </summary>
        protected override void OnConnected()
        {
            base.OnConnected();

            // Get the SimObject data for this aircraft.
            GetSimObjectData();
        }

        /// <summary>
        /// Register for any events that need to be listened to.
        /// </summary>
        protected override void SimConnectEventRegistration()
        {
            // Register for the base classes desired events.
            base.SimConnectEventRegistration();

            // Register for the sim connect events we care about.
            simulation_connection.OnRecvSimobjectDataBytype += new SimConnect.RecvSimobjectDataBytypeEventHandler(SimulationConnectionSimobjectDataByType);
        }

        /// <summary>
        /// Define the data structures that we need for talking to Prepar3D.
        /// </summary>
        protected override void DefineDataStructures()
        {
            // Define any base class data structures that are needed.
            base.DefineDataStructures();

            DefineSimObjectStructure();
            DefineHealthStatusStructure();
        }

        /// <summary>
        /// Unloads the current aircraft if it exists.
        /// </summary>
        private void UnloadCurrentObjectData()
        {
            // Only do this if there is an aircraft already.
            if (current_object != null)
            {
                try
                {
                    current_object = null;
                }
                catch (Exception ex)
                {
                }
            }
        }
    }
}
```

```

        SendError(ex.Message);
    }
}

/// <summary>
/// Defines the SimObject data structure that will be exchanged between this app and the Prepar3D simulation.
/// </summary>
private void DefineSimObjectStructure()
{
    // Define the SimObject data structure. These MUST be added in the EXACT same order as they are listed in the structure.
    simulation_connection.AddaDataDefinition(StructureType.SimObject, "Is User Sim", "Bool", SIMCONNECT_DATATYPE.FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
    simulation_connection.AddaDataDefinition(StructureType.SimObject, "Is Alive", "Bool", SIMCONNECT_DATATYPE.FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
    simulation_connection.AddaDataDefinition(StructureType.SimObject, "Health Points", "Number", SIMCONNECT_DATATYPE.INT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
    simulation_connection.AddaDataDefinition(StructureType.SimObject, "Title", null, SIMCONNECT_DATATYPE.STRING256, 0.0f, SimConnect.SIMCONNECT_UNUSED);

    // IMPORTANT: register it with the simconnect managed wrapper marshaller
    // if you skip this step, you will only receive a uint in the .dwData field.
    simulation_connection.RegisterDataDefineStruct<SimObjectData>(StructureType.SimObject);
}

/// <summary>
/// Defines the HealthStatus data structure.
/// </summary>
private void DefineHealthStatusStructure()
{
    // Define the health status data structure. These MUST be added in the EXACT same order as they are listed in the structure.
    simulation_connection.AddaDataDefinition(StructureType.HealthStatus, "Health Points", "Number",
                                             SIMCONNECT_DATATYPE.INT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);

    // IMPORTANT: register it with the simconnect managed wrapper marshaller
    // if you skip this step, you will only receive a uint in the .dwData field.
    simulation_connection.RegisterDataDefineStruct<HealthStatus>(StructureType.HealthStatus);
}

/// <summary>
/// Turn a sim object into an aircraft class.
/// </summary>
/// <param name="simObject"></param>
/// <returns></returns>
private SimObject SimObjectToClass(SimObjectData simObject, uint objectID)
{
    SimObject data;

    // Create the new aircraft.
    data = new SimObject();

    // Set the values of the SimObject based on the sim object's data.
    data.ObjectID = objectID;
    data.IsUserSim = (simObject.IsUserSim == 0) ? false : true;
    data.IsAlive = (simObject.IsAlive == 0) ? false : true;
    data.HealthPoints = simObject.HealthPoints;
    data.Title = simObject.Title;

    // Return the SimObject.
    return data;
}

/// <summary>
/// Turn a sim object into an aircraft class.
/// </summary>
/// <param name="simObject"></param>
/// <returns></returns>
private HealthUpdate HealthStatusToClass(HealthStatus status, uint objectID)
{
    HealthUpdate update;

    // Create the new aircraft.
    update = new HealthUpdate();

    // Set the values of the SimObject based on the sim object's data.
    update.ObjectID = objectID;
    update.HealthPoints = status.LifePoints;

    // Return the SimObject.
    return update;
}

/// <summary>
/// This is where we will be receiving SimObject information.
/// </summary>
/// <param name="sender"></param>
/// <param name="data"></param>
private void SimulationConnectionSimObjectDataByType(SimConnect sender, SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE data)
{
    // Determine what type of SimObject data we received by checking the data ID against our DataIdentifier Enum.
    switch ((DataIdentifier)data.dwRequestId)
    {
        // Handle SimObject data.
        case DataIdentifier.RetrieveSimObjects:
        {
            SimObjectData simObject;

            // Convert the SimObject to a SimObject and let whoever cares know that there is a SimObject.
            simObject = (SimObjectData)data.dwData[0];
            SimObjectDataRequests.SafeInvoke(this, new GenericEventArgs<SimObject>(SimObjectToClass(simObject, data.dwObjectID)));
            break;
        }

        case DataIdentifier.RetrieveHealthInfo:
        {
            HealthStatus status;

            // Convert this to a HealthStatusUpdate.
            status = (HealthStatus)data.dwData[0];
            HealthStatusRequests.SafeInvoke(this, new GenericEventArgs<HealthUpdate>(HealthStatusToClass(status, data.dwObjectID)));
            break;
        }

        // No idea what this is.
        default:
        {
            SendError("Unknown request ID: " + data.dwRequestId.ToString());
            break;
        }
    }
}
#endregion
}

```

```

SimConnector.cs
using System;
using System.Collections.Generic;
using System.Threading;

using LockheedMartin.Prepar3D.SimConnect;

namespace AIDestroyer
{
    public class SimConnector
    {
        #region Events
        /// <summary>
        /// Fires off when this connects to the simulation.
        /// </summary>
        public event EventHandler<EventArgs> Connected;

        /// <summary>
        /// Fires off when this disconnects from the simulation.
        /// </summary>
        public event EventHandler<EventArgs> Disconnected;

        /// <summary>
        /// Fires off when an error has occurred.
        /// </summary>
        public event EventHandler<MessageEventArgs> Error;
        #endregion

        #region Fields
        /// <summary>

```

```

    ///> The name for this sim connect connection.
    ///> </summary>
    private string connection_name;

    ///> <summary>
    ///> The connection to the Prepar3D simulator.
    ///> </summary>
    protected SimConnect simulation_connection;

    ///> <summary>
    ///> This helps time the messages from sim connect so we can read when
    ///> the message thread is no longer blocked. This is what blocks and waits for a signal.
    ///> </summary>
    private EventWaitHandle message_pump;

    ///> <summary>
    ///> The thread that will constantly run and pump messages for us.
    ///> </summary>
    private Thread message_thread;
    #endregion

    #region Properties
    #endregion

    #region Constructors
    ///> <summary>
    ///> Create a new simulation connection.
    ///> </summary>
    ///> <param name="connectionName">The name for this connection.</param>
    public SimConnector(string connectionName)
    {
        // Store the name for this simulation connection.
        connection_name = connectionName;

        // Set the simulation connection members to their default values.
        simulation_connection = null;
        message_pump = null;
        message_thread = null;
    }
    #endregion

    #region Methods
    ///> <summary>
    ///> Connects to the Prepar3D simulation.
    ///> </summary>
    public void Connect()
    {
        // Try to start the SimConnect connection.
        try
        {
            // If we are already connected, then don't do anything.
            if (!IsConnectionActive())
            {
                // Create a simulation connection. Since we are using WPF we should create a
                // threaded event message pump that can easily get the data coming from SimConnect.
                message_pump = new EventWaitHandle(false, EventResetMode.AutoReset);
                simulation_connection = new SimConnect(connection_name, IntPtr.Zero, 0, message_pump, 0);

                // Register for the events and data structures we care about.
                SimConnectEventRegistration();
                DefineDataStructures();

                // Now create a thread to handle to handle the message processing and
                // run it as a background thread.
                message_thread = new Thread(new ThreadStart(MessageProcessor));
                message_thread.IsBackground = true;
                message_thread.Start();
            }
        }
        catch (Exception ex)
        {
            // Throw an Error explaining what happened.
            SendError(ex.Message);
        }
    }

    ///> <summary>
    ///> Disconnects from the Prepa3D simulation.
    ///> </summary>
    public void Disconnect()
    {
        // Stop the message pump thread.
        if (IsConnectionActive())
        {
            // Handle disconnecting properly.
            OnDisconnected();
        }
    }

    ///> <summary>
    ///> Register for the SimConnect events we care about.
    ///> </summary>
    protected virtual void SimConnectEventRegistration()
    {
        // Register for the events we care about.
        simulation_connection.OnRecvOpen += new SimConnect.RecvOpenEventHandler(SimulationConnectionOpen);
        simulation_connection.OnRecvQuit += new SimConnect.RecvQuitEventHandler(SimulationConnectionClosed);
        simulation_connection.OnRecvException += new SimConnect.RecvExceptionEventHandler(SimulationConnectionException);
    }

    ///> <summary>
    ///> define the data structures that we care about.
    ///> </summary>
    protected virtual void DefineDataStructures()
    {
    }

    ///> <summary>
    ///> Called when this connector has connected with Prepar3D.
    ///> </summary>
    protected virtual void OnConnected()
    {
        // Send the event that says we have connected to the simulation.
        Connected.SafeInvoke(this, EventArgs.Empty);
    }

    ///> <summary>
    ///> Called when this connector has been disconnected from Prepar3D.
    ///> Handle the base call after the derived classes stuff if you need simulation_connection to be valid.
    ///> </summary>
    protected virtual void OnDisconnected()
    {
        // Stop the message pump thread.
        message_thread.Abort();

        // Since we are no longer processing messages, close the connection to the simulation.
        simulation_connection.Dispose();
        message_pump.Dispose();

        // Reset the state of the data model.
        Reset();

        // Send the disconnected event.
        Disconnected.SafeInvoke(this, EventArgs.Empty);
    }

    ///> <summary>
    ///> Allows derived classes to raise the error event.
    ///> </summary>
    ///> <param name="message">The message to send.</param>
    protected void SendError(string message)
    {
        Error.SafeInvoke(this, new MessageEventArgs(message));
    }

    ///> <summary>
    ///> Resets the connection data.
    ///> </summary>
    private void Reset()
    {
        simulation_connection = null;
        message_pump = null;
        message_thread = null;
    }

```

```

/// <summary>
/// Returns whether or not the simulation is currently connected.
/// </summary>
/// <returns><c>true</c> if the simulation is connected; otherwise <c>false</c>.</returns>
private bool IsConnectionActive()
{
    return (simulation_connection != null);
}

/// <summary>
/// Handles when exceptions occur from SimConnect.
/// </summary>
/// <param name="sender"></param>
/// <param name="data"></param>
private void SimulationConnectionException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    // Show the exception to the user.
    SendError("Exception received: " + data.dwException.ToString());
}

/// <summary>
/// Handles the simulation Quit event. This is fired off if Prepar3D is closed.
/// </summary>
/// <param name="sender"></param>
/// <param name="data"></param>
private void SimulationConnectionClosed(SimConnect sender, SIMCONNECT_RECV data)
{
    // Disconnect from the simulation.
    Disconnect();
}

/// <summary>
/// Handles the simulation Open event. This is fired off when we first connect to the simulator.
/// </summary>
/// <param name="sender"></param>
/// <param name="data"></param>
private void SimulationConnectionOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
{
    OnConnected();
}

/// <summary>
/// The message processing function.
/// </summary>
private void MessageProcessor()
{
    // Process this thread unless we are aborted.
    while (true)
    {
        // Block until a message comes through.
        message_pump.WaitOne();

        // Process the simconnect message.
        if (simulation_connection != null)
        {
            try
            {
                // Try to receive a message.
                simulation_connection.ReceiveMessage();
            }
            catch (Exception ex)
            {
                // Throw an error with the exceptions message.
                SendError(ex.Message);
            }
        }
    }
}
#endregion
}

```

SimConnectModel.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Threading;

using LockheedMartin.Prepar3D.SimConnect;

namespace AIDestroyer
{
    public class SimConnectModel : INotifyPropertyChanged
    {
        #region Constants
        /// <summary>
        /// The connected status string.
        /// </summary>
        private const string CONNECTED_STATUS = "Connected";

        /// <summary>
        /// The disconnected status string.
        /// </summary>
        private const string DISCONNECTED_STATUS = "Disconnected";

        /// <summary>
        /// The error status string.
        /// </summary>
        private const string ERROR_STATUS = "Error";
        #endregion

        #region Events
        /// <summary>
        /// Fired off when a property is changed.
        /// This allows the data model to be bound to the view.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        /// <summary>
        /// An event that is raised when errors in the data model occur.
        /// </summary>
        public event EventHandler<EventArgs> Error;
        #endregion

        #region Fields
        /// <summary>
        /// The status of the current simulation connection.
        /// </summary>
        private string connection_status;
        #endregion

        #region Properties
        /// <summary>
        /// Get the status of the current simulation connection.
        /// </summary>
        public string Status
        {
            get
            {
                return connection_status;
            }

            private set
            {
                connection_status = value;
                // Mark that this property has changed so any bound items can be updated.
                OnPropertyChanged("Status");
            }
        }
        /// <summary>
        /// Gets or sets the simulation connection currently being used.
        /// If you derive from this class then assign this to the
        /// base class for class derivative you are using.
        /// </summary>
        protected SimConnector SimulationConnection
        { get; set; }
        #endregion
    }
}

```

```


/// Creates the data model for this SDK sample.

public SimConnectModel()
{
    SimulationConnection = null;
    Status = DISCONNECTED_STATUS;
}
#endregion

#region Methods

/// Initialize the data model.

public virtual void Init()
{
    // Register for the events about the simulation connection.
    if (SimulationConnection != null)
    {
        SimulationConnection.Connected += new EventHandler<EventArgs>(SimConnectorConnected);
        SimulationConnection.Disconnected += new EventHandler<EventArgs>(SimConnectorDisconnected);
        SimulationConnection.Error += new EventHandler<MessageEventArgs>(SimConnectorError);
    }
}


/// Connect to the Prepar3D simulation.

public void ConnectToSimulation()
{
    if (SimulationConnection != null)
    {
        SimulationConnection.Connect();
    }
}


/// Disconnect from the Prepar3D simulation.

public void DisconnectFromSimulation()
{
    if (SimulationConnection != null)
    {
        SimulationConnection.Disconnect();
    }
}


/// Notify any interested parties that a property has changed.

protected void OnPropertyChanged(string name)
{
    PropertyChangedEventHandler handler;

    // Needs a safe invoke that does this check for you. :-(

    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(name));
    }
}


/// Sends out an error event.

protected void SendError(string message)
{
    // Send an error event.
    Error.SafeInvoke(this, new MessageEventArgs(message));
}


/// Handle the event of the SimConnector connecting to Prepar3D.

protected virtual void SimConnectorConnected(object sender, EventArgs args)
{
    Status = CONNECTED_STATUS;
}


/// Handle the event of the SimConnector disconnecting to Prepar3D.

protected virtual void SimConnectorDisconnected(object sender, EventArgs args)
{
    Status = DISCONNECTED_STATUS;
}


/// Handle the event of the SimConnector having an error.

protected virtual void SimConnectorError(object sender, MessageEventArgs args)
{
    Status = ERROR_STATUS;

    // Forward on this error to any interested parties.
    // I do this instead of send error because I don't have to create another MessageEventArgs then.
    Error.SafeInvoke(this, args);
}
#endregion
}

```

```

DestroyerModel.cs
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Threading;
using System.Windows.Threading;
namespace AIDestroyer
{
    public class DestroyerModel : SimConnectModel
    {
        #region Events
        /// <summary>
        /// Defines a function that can be invoked for adding a SimObject to the SimObjectList.
        /// This is used to put the call on the right thread.
        /// </summary>
        /// <param name="simObject"></param>
        private delegate void AddSimObjectHandler(SimObject simObject);
        #endregion

        #region Fields
        /// <summary>
        /// The connection to the simulation.
        /// </summary>
        private DestroyerConnector simulation_connection;
        /// <summary>
        /// The selected object.
        /// </summary>
        private SimObject selected_object;
        #endregion

        #region Properties
        /// <summary>
        /// A list of all the SimObjects in the simulation that are within range of the User's simObject.
        /// </summary>
        public ObservableCollection<SimObject> SimObjectList
        { get; private set; }
    }
}

```

```

    ///<summary>
    ///<summary>The currently selected object.
    ///</summary>
    public SimObject SelectedObject
    {
        get
        {
            return selected_object;
        }

        set
        {
            selected_object = value;
            simulation_connection.SetCurrentObject(selected_object);

            // Mark that some of the properties of this class have changed.
            OnPropertyChanged("SelectedObject");
            OnPropertyChanged("IsSimObjectAlive");
            OnPropertyChanged("IsSimObjectSelected");
        }
    }

    ///<summary>
    ///<summary>Returns if the currently selected SimObject is alive.
    ///</summary>
    public bool IsSimObjectAlive
    {
        get
        {
            if (selected_object != null)
            {
                return selected_object.IsAlive;
            }

            // Default to the SimObject being dead.
            return false;
        }

        set
        {
            selected_object.IsAlive = value;

            if (selected_object.IsAlive)
            {
                simulation_connection.SetHealth(1);
            }
            else
            {
                simulation_connection.SetHealth(0);
            }

            OnPropertyChanged("IsSimObjectAlive");
        }
    }

    ///<summary>
    ///<summary>Returns whether or not a SimObject is currently selected.
    ///</summary>
    public bool IsSimObjectSelected
    {
        get
        {
            return selected_object != null;
        }
    }
}

#region Constructors
///<summary>
///<summary>Create a new Weapon Data Model.
///</summary>
public DestroyerModel()
{
    // Create a new simulation connection.
    simulation_connection = new DestroyerConnector("AI Destroyer");
    SimulationConnection = simulation_connection;

    // Now initialize the observable collection.
    SimObjectList = new ObservableCollection<SimObject>();
}
#endregion

#region Methods
///<summary>
///<summary>Initialize this data model.
///</summary>
public override void Init()
{
    base.Init();

    // Register to receive the sim object data requests from the weapon connector.
    simulation_connection.SimObjectDataRequests += new EventHandler<GenericEventArgs<SimObject>>(SimulationConnectionSimObjectDataRequests);
    simulation_connection.HealthStatusRequests += new EventHandler<GenericEventArgs<HealthUpdate>>(SimulationConnectionHealthStatusRequests);
}

///<summary>
///<summary>Update the list of SimObject we know about that have weapons.
///</summary>
public void UpdateSimList()
{
    // Clear the SimObjectlist.
    SimObjectList.Clear();

    // Now just get all the SimObject around the user again.
    simulation_connection.GetSimObjectData();
}

///<summary>
///<summary>Selects the given SimObject to be used as the main SimObject.
///</summary>
///<param name="simObject"></param>
public void SelectSimObject(SimObject simObject)
{
    // Set the selected object.
    SelectedObject = simObject;
}

///<summary>
///<summary>Add an simObject to the sim object list.
///</summary>
///<param name="simObject"></param>
private void AddSimObject(SimObject simObject)
{
    // I only care about SimObject that are not the user and I don't want any repeats.
    if (!simObject.IsUserSim && !SimObjectList.Contains(simObject))
    {
        SimObjectList.Add(simObject);
    }
}

///<summary>
///<summary>Receives the SimObjectDataRequest and adds the simObject to the SimObjectList.
///</summary>
///<param name="sender"></param>
///<param name="args"></param>
private void SimulationConnectionSimObjectDataRequests(object sender, GenericEventArgs<SimObject> args)
{
    AddSimObjectHandler aircraftCreation;

    // Need to do this to get the add call on the right thread.
    aircraftCreation = AddSimObject;

    // Should make a multithreaded ObservableCollection instead.....
    System.Windows.Application.Current.Dispatcher.Invoke(aircraftCreation, args.Value);
}

///<summary>
///<summary>Receives the SimObjectDataRequest and adds the simObject to the SimObjectList.
///</summary>
///<param name="sender"></param>
///<param name="args"></param>
private void SimulationConnectionHealthStatusRequests(object sender, GenericEventArgs<HealthUpdate> args)
{
}

```

```
{  
    // Go through and find out what SimObject this HealthStatus is for and update it.  
    foreach (SimObject simObject in SimObjectList)  
    {  
        if (simObject.ObjectID == args.Value.ObjectID)  
        {  
            simObject.HealthPoints = args.Value.HealthPoints;  
        }  
    }  
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

AI_Waypoints.cs

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
///
/// Managed AI Waypoints sample
///
// Ensure that your aircraft is at the start of the main runway at Sea-Tac airport (KSEA).
///
// Click on Connect to try and connect to a running version of Prepar3D
// Click on Create AI objects once to create two aircraft and a fuel truck
// Click on Send AI Waypoints once to send waypoints to the two AI objects
// Watch the Maule M7 and fuel truck go into motion
// Click on Send toggle light request to see the Maule turn lights on/off
// Click on Disconnect to close the connection, and then you will
// be able to click on Connect and restart the process
///

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// Add these two statements to all SimConnect clients
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;

namespace Managed_AI_Waypoints
{
    public partial class AI_Waypoints : Form
    {

        // User-defined win32 event
        const int WM_USER_SIMCONNECT = 0x0402;

        // SimConnect object
        SimConnect simconnect = null;

        // Object IDs
        uint MauleID = 0;
        uint TruckID = 0;
        uint MooneyBravoID = 0;

        enum DEFINITIONS
        {
            MauleWaypoints,
            FuelTruckWaypoints,
            MauleLights,
        }

        enum DATA_REQUESTS
        {
            REQUEST_MOONEY,
            REQUEST_MAULE,
            REQUEST_TRUCK,
        };

        enum EVENTS
        {
            PAUSED,
            SEND_UNPAUSE,
        };

        enum GROUPID
        {
            FLAG = 2000000000,
        };

        // this is how you declare a data structure so that
        // simconnect knows how to fill it/read it.
        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
        struct MauleLights
        {
            public uint lightStates;
            // Add more data here if necessary
        };

        // Declare the actual structure
        MauleLights mauleLights;

        public AI_Waypoints()
        {
            InitializeComponent();
            setButtons(true, false, false, false, false);
            mauleLights.lightStates = 0;
        }

        // Simconnect client will send a win32 message when there is
        // a packet to process. ReceiveMessage must be called to
        // trigger the events. This model keeps simconnect processing on the main thread.
        protected override void DefWndProc(ref Message m)
        {
            if (m.Msg == WM_USER_SIMCONNECT)
            {
                if (simconnect != null)
                {
                    simconnect.ReceiveMessage();
                }
            }
            else
            {
                base.DefWndProc(ref m);
            }
        }

        private void setButtons(bool bConnect, bool bCreate, bool bSend, bool bLights, bool bDisconnect)
        {
            buttonConnect.Enabled = bConnect;
            buttonCreateAIObjects.Enabled = bCreate;
            buttonSendWaypoints.Enabled = bSend;
            buttonLights.Enabled = bLights;
            buttonDisconnect.Enabled = bDisconnect;
        }

        private void closeConnection()
        {
            if (simconnect != null)
            {
                // Dispose serves the same purpose as SimConnect_Close()
                simconnect.Dispose();
                simconnect = null;
                mauleLights.lightStates = 0;
                displayText("Connection closed");
            }
        }

        // Set up the SimConnect event handlers
        private void initComms()
        {
            try
            {
                // listen to connect and quit msgs
                simconnect.OnRecvOpen += new SimConnect.RecvOpenEventHandler(simconnect_OnRecvOpen);
                simconnect.OnRecvQuit += new SimConnect.RecvQuitEventHandler(simconnect_OnRecvQuit);

                // listen to exceptions
                simconnect.OnRecvException += new SimConnect.RecvExceptionEventHandler(simconnect_OnRecvException);
            }
        }
    }
}
```

```

// catch the assigned object IDs
simconnect.OnRecvAssignedObjectid += new SimConnect.RecvAssignedObjectIdEventHandler(simconnect_OnRecvAssignedObjectId);

// set up the data definiton for the Maule lights
simconnect.AddToDataDefinition(DEFINITIONS.MauleLights, "LIGHT STATES", "mask", SIMCONNECT_DATATYPE.INT32, 0.Of, SimConnect.SIMCONNECT_UNUSED);

// IMPORTANT: register it with the simconnect managed wrapper marshaller
// if you skip this step, you will only receive a uint in the .dwData field.
simconnect.RegisterDataDefineStruct<MauleLights>(DEFINITIONS.MauleLights);

// Subscribe to system event Pause

simconnect.SubscribeToSystemEvent(EVENTS.PAUSED, "Pause");
simconnect.OnRecvEvent += new SimConnect.RecvEventEventHandler(simconnect_OnRecvEvent);

// Map an event to the EventID: PAUSE_OFF
simconnect.MapClientEventToSimEvent(EVENTS.SEND_UNPAUSE, "PAUSE_OFF");

}

catch (COMException ex)
{
    displayText(ex.Message);
}

// The simulation will pause each time a key is selected in the addon, so unpause the sim each time this happens

void simconnect_OnRecvEvent(SimConnect sender, SIMCONNECT_RECV_EVENT data)
{
    displayText("Pause event received");

    switch ((EVENTS)data.uEventID)
    {
        case EVENTS.PAUSED:
            simconnect.TransmitClientEvent((uint) SimConnect.SIMCONNECT_OBJECT_ID_USER, EVENTS.SEND_UNPAUSE, (uint) 0, GROUPID.FLAG, SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
            displayText("Unpause request sent...");
            break;
    }
}

void simconnect_OnRecvAssignedObjectid(SimConnect sender, SIMCONNECT_RECV_ASSIGNED_OBJECT_ID data)
{
    switch ((DATA_REQUESTS)data.dwRequestId)
    {
        case DATA_REQUESTS.REQUEST_MOONEY:
            MooneyBravoID = (uint) (DATA_REQUESTS)data.dwObjectID;
            displayText("Received Mooney Bravo ID");
            break;

        case DATA_REQUESTS.REQUEST_MAULE:
            MauleID = (uint) (DATA_REQUESTS)data.dwObjectID;
            displayText("Received Maule M7 ID");
            break;

        case DATA_REQUESTS.REQUEST_TRUCK:
            TruckID = (uint) (DATA_REQUESTS)data.dwObjectID;
            displayText("Received Truck ID");
            break;

        default:
            displayText("Unknown Request ID received: " + (DATA_REQUESTS)data.dwRequestId);
            break;
    }
}

void simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
{
    displayText("Connected to Prepar3D");
}

// The case where the user closes Prepar3D
void simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
{
    displayText("Prepar3D has exited");
    closeConnection();
}

void simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    displayText("Exception received: " + data.dwException);
}

// The case where the user closes the client
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    closeConnection();
}

private void buttonConnect_Click(object sender, EventArgs e)
{
    if (simconnect == null)
    {
        try
        {
            // the constructor is similar to SimConnect_Open in the native API
            simconnect = new SimConnect("Managed AI Waypoints", this.Handle, WM_USER_SIMCONNECT, null, 0);

            setButtons(false, true, false, false, true);

            initComms();

        }
        catch
        {
            displayText("Unable to connect to Prepar3D!");
        }
    }
    else
    {
        displayText("Error - try again!");
        closeConnection();

        setButtons(true, false, false, false, false);
    }
}

private void buttonDisconnect_Click(object sender, EventArgs e)
{
    closeConnection();
    setButtons(true, false, false, false, false);
}

private void buttonLights_Click(object sender, EventArgs e)
{
    if (MauleID != 0)
    {
        // Toggle the lights
        mauleLights.lightStates = (mauleLights.lightStates == 0) ? uint.MaxValue : 0;

        simconnect.SetDataOnSimObject(DEFINITIONS.MauleLights, MauleID, 0, mauleLights.lightStates);

        if (mauleLights.lightStates != 0)
            displayText("Lights on requested...");
        else
            displayText("Lights off requested...");
    }
    else
        displayText("Maule M7 ID not set!");
}

private void buttonSendWaypoints_Click(object sender, EventArgs e)
{
    if (MauleID != 0 && TruckID != 0)
    {
        SIMCONNECT_DATA_WAYPOINT[] wp = new SIMCONNECT_DATA_WAYPOINT[3];
        SIMCONNECT_DATA_WAYPOINT[] ft = new SIMCONNECT_DATA_WAYPOINT[2];

        simconnect.AddToDataDefinition(DEFINITIONS.MauleWaypoints, "AI WAYPOINT LIST", "number", SIMCONNECT_DATATYPE.WAYPOINT, 0.Of, SimConnect.SIMCONNECT_UNUSED);
        simconnect.AddToDataDefinition(DEFINITIONS.FuelTruckWaypoints, "AI WAYPOINT LIST", "number", SIMCONNECT_DATATYPE.WAYPOINT, 0.Of, SimConnect.SIMCONNECT_UNUSED);

        // Maule aircraft should fly in circles across the North end of the runway
    }
}

```

```

wp[0].Flags = (uint)SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED;
wp[0].Altitude = 800;
wp[0].Latitude = 47 + (27.79 / 60);
wp[0].Longitude = -122 - (18.46 / 60);
wp[0].ktsSpeed = 100;

wp[1].Flags = (uint)SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED;
wp[1].Altitude = 800;
wp[1].Latitude = 47 + (27.79 / 60);
wp[1].Longitude = -122 - (17.37 / 60);
wp[1].ktsSpeed = 100;

wp[2].Flags = (uint)(SIMCONNECT_WAYPOINT_FLAGS.WRAP_TO_FIRST | SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED);
wp[2].Altitude = 800;
wp[2].Latitude = 47 + (27.79 / 60);
wp[2].Longitude = -122 - (19.92 / 60);
wp[2].ktsSpeed = 100;

// Create a polymorphic array
Object[] objv1 = new Object[wp.Length];
wp.CopyTo(objv1, 0);

// Send the three waypoints to the Maule
simconnect.SetDataOnSimObject(DEFINITIONS.MauleWaypoints, MauleID, 0, objv1);

// Truck goes down the runway
ft[0].Flags = (uint)SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED;
ft[0].Altitude = 433;
ft[0].Latitude = 47 + (25.93 / 60);
ft[0].Longitude = -122 - (18.46 / 60);
ft[0].ktsSpeed = 75;

ft[1].Flags = (uint)(SIMCONNECT_WAYPOINT_FLAGS.WRAP_TO_FIRST | SIMCONNECT_WAYPOINT_FLAGS.SPEED_REQUESTED);
ft[1].Altitude = 433;
ft[1].Latitude = 47 + (26.25 / 60);
ft[1].Longitude = -122 - (18.46 / 60);
ft[1].ktsSpeed = 55;

// Create a polymorphic array
Object[] objv2 = new Object[ft.Length];
ft.CopyTo(objv2, 0);

// Send the waypoints to the fuel truck
simconnect.SetDataOnSimObject(DEFINITIONS.FuelTruckWaypoints, TruckID, 0, objv2);

displayText("Waypoint lists sent...");
setButtons(false, false, false, false, true);
} else
    displayText("Maule M7 or Truck IDs not set!");
}

private void buttonCreateAIObjects_Click(object sender, EventArgs e)
{
    SIMCONNECT_DATA_INITPOSITION Init;

    // Add a parked museum aircraft, just west of the runway
    Init.Altitude = 433.0;           // Altitude of Sea-tac is 433 feet
    Init.Latitude = 47 + (25.97 / 60); // Convert from 47 25.97 N
    Init.Longitude = -122 - (18.51 / 60); // Convert from 122 18.51 W
    Init.Pitch = 0.0;
    Init.Bank = 0.0;
    Init.Heading = 90.0;
    Init.OnGround = 1;
    Init.Airspeed = 0;
    simconnect.AICreateSimulatedObject("Mooney Bravo", Init, DATA_REQUESTS.REQUEST_MOONEY);

    // Initialize Maule M7 aircraft just in front of user aircraft, at 47 25.89 N, 122 18.48 W
    Init.Altitude = 433.0;           // Altitude of Sea-tac is 433 feet
    Init.Latitude = 47 + (25.91 / 60); // Convert from 47 25.90 N
    Init.Longitude = -122 - (18.48 / 60); // Convert from 122 18.48 W
    Init.Pitch = 0.0;
    Init.Bank = 0.0;
    Init.Heading = 360.0;
    Init.OnGround = 1;
    Init.Airspeed = 0;
    simconnect.AICreateNonATCAircraft("Maule M7 260C paint2", "N1001", Init, DATA_REQUESTS.REQUEST_MAULE);

    // Initialize truck just in front of user aircraft
    // User aircraft is at 47 25.89 N, 122 18.48 W
    Init.Altitude = 433.0;           // Altitude of Sea-tac is 433 feet
    Init.Latitude = 47 + (25.91 / 60); // Convert from 47 25.90 N
    Init.Longitude = -122 - (18.47 / 60); // Convert from 122 18.48 W
    Init.Pitch = 0.0;
    Init.Bank = 0.0;
    Init.Heading = 360.0;
    Init.OnGround = 1;
    Init.Airspeed = 0;
    simconnect.AICreateSimulatedObject("VEH_jetTruck", Init, DATA_REQUESTS.REQUEST_TRUCK);

    displayText("Request to create objects sent...");
    setButtons(false, false, true, true, true);
}

// Response number
int response = 1;

// Output text - display a maximum of 10 lines
string output = "\n\n\n\n\n\n\n\n\n\n";
}

void displayText(string s)
{
    // remove first string from output
    output = output.Substring(output.IndexOf("\n") + 1);

    // add the new string
    output += "\n" + response++ + ":" + s;

    // display it
    richResponse.Text = output;
}
}

// End of sample

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

ManagedChangeVehicle.cs

```
///-
/// Copyright 2015 Lockheed Martin Corporation
/// All Rights Reserved
///
/// Description: Managed sample code for changing the vehicle Prepar3D is
/// running through SimConnect.
///
///

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Threading;

using LockheedMartin.Prepar3D.SimConnect;

namespace ManagedChangeVehicle
{
    class ManagedChangeVehicle
    {
        static void Main(string[] args)
        {
            try
            {
                SimConnect simulation_connection = new SimConnect("Managed Change Vehicle", IntPtr.Zero, 0,
                    null, 0);

                // Sending the title of the Vehicle
                simulation_connection.ChangeVehicle("Mooney Bravo Retro");

                simulation_connection.Dispose();
                simulation_connection = null;
            }
            catch (Exception ex)
            {
                // We were unable to connect so let the user know why.
                System.Console.WriteLine("Sim Connect unable to connect to Prepar3D!\n\n{0}\n\n{1}",
                    ex.Message, ex.StackTrace);
            }
        }
    }
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

Client Event.cs

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//
// Managed Client Event sample
//
// Respond to Flaps up and down (F5, F6, F7 and F8 keys)
// Respond to Pitot switch (Shift-H)
//

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// Add these two statements to all SimConnect clients
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;

namespace Managed_Client_Event
{
    public partial class Form1 : Form
    {

        // User-defined win32 event
        const int WM_USER_SIMCONNECT = 0x0402;

        // SimConnect object
        SimConnect simconnect = null;

        enum EVENTS
        {
            PITOT_TOGGLE,
            FLAPS_INC,
            FLAPS_DEC,
            FLAPS_UP,
            FLAPS_DOWN,
        };

        enum NOTIFICATION_GROUPS
        {
            GROUP0,
        }

        public Form1()
        {
            InitializeComponent();

            setButtons(true, false);
        }

        // Simconnect client will send a win32 message when there is
        // a packet to process. ReceiveMessage must be called to
        // trigger the events. This model keeps simconnect processing on the main thread.

        protected override void DefWndProc(ref Message m)
        {
            if (m.Msg == WM_USER_SIMCONNECT)
            {
                if (simconnect != null)
                {
                    simconnect.ReceiveMessage();
                }
            }
            else
            {
                base.DefWndProc(ref m);
            }
        }

        private void setButtons(bool bConnect, bool bDisconnect)
        {
            buttonConnect.Enabled = bConnect;
            buttonDisconnect.Enabled = bDisconnect;
        }

        private void closeConnection()
        {
            if (simconnect != null)
            {
                // Dispose serves the same purpose as SimConnect_Close()
                simconnect.Dispose();
                simconnect = null;
                displayText("Connection closed");
            }
        }

        // Set up all the SimConnect related event handlers
        private void initClientEvent()
        {
            try
            {
                // listen to connect and quit msgs
                simconnect.OnRecvOpen += new SimConnect.RecvOpenEventHandler(simconnect_OnRecvOpen);
                simconnect.OnRecvQuit += new SimConnect.RecvQuitEventHandler(simconnect_OnRecvQuit);
            }
        }
    }
}
```

```

        // listen to exceptions
        simconnect.OnRecvException += new SimConnect.RecvExceptionHandler(simconnect_OnRecvException);

        // listen to events
        simconnect.OnRecvEvent += new SimConnect.RecvEventEventHandler(simconnect_OnRecvEvent);

        // subscribe to pitot heat switch toggle
        simconnect.MapClientEventToSimEvent(EVENTS.PITOT_TOGGLE, "PITOT_HEAT_TOGGLE");
        simconnect.AddClientEventToNotificationGroup(NOTIFICATION_GROUPS.GROUP0, EVENTS.PITOT_TOGGLE, false);

        // subscribe to all four flaps controls
        simconnect.MapClientEventToSimEvent(EVENTS.FLAPS_UP, "FLAPS_UP");
        simconnect.AddClientEventToNotificationGroup(NOTIFICATION_GROUPS.GROUP0, EVENTS.FLAPS_UP, false);
        simconnect.MapClientEventToSimEvent(EVENTS.FLAPS_DOWN, "FLAPS_DOWN");
        simconnect.AddClientEventToNotificationGroup(NOTIFICATION_GROUPS.GROUP0, EVENTS.FLAPS_DOWN, false);
        simconnect.MapClientEventToSimEvent(EVENTS.FLAPS_INC, "FLAPS_INCR");
        simconnect.AddClientEventToNotificationGroup(NOTIFICATION_GROUPS.GROUP0, EVENTS.FLAPS_INC, false);
        simconnect.MapClientEventToSimEvent(EVENTS.FLAPS_DEC, "FLAPS_DECR");
        simconnect.AddClientEventToNotificationGroup(NOTIFICATION_GROUPS.GROUP0, EVENTS.FLAPS_DEC, false);

        // set the group priority
        simconnect.SetNotificationGroupPriority(NOTIFICATION_GROUPS.GROUP0, SimConnect.SIMCONNECT_GROUP_PRIORITY_HIGHEST);
    }

    catch (COMException ex)
    {
        displayText(ex.Message);
    }
}

void simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
{
    displayText("Connected to Prepar3D");
}

// The case where the user closes Prepar3D
void simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
{
    displayText("Prepar3D has exited");
    closeConnection();
}

void simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    displayText("Exception received: " + data.dwException);
}

void simconnect_OnRecvEvent(SimConnect sender, SIMCONNECT_RECV_EVENT recEvent)
{
    switch (recEvent.uEventID)
    {
        case (uint)EVENTS.PITOT_TOGGLE:
            displayText("PITOT switched");
            break;

        case (uint)EVENTS.FLAPS_UP:
            displayText("Flaps Up");
            break;

        case (uint)EVENTS.FLAPS_DOWN:
            displayText("Flaps Down");
            break;

        case (uint)EVENTS.FLAPS_INC:
            displayText("Flaps Inc");
            break;

        case (uint)EVENTS.FLAPS_DEC:
            displayText("Flaps Dec");
            break;
    }
}

// The case where the user closes the client
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    closeConnection();
}

private void buttonConnect_Click(object sender, EventArgs e)
{
    if (simconnect == null)
    {
        try
        {
            // the constructor is similar to SimConnect_Open in the native API
            simconnect = new SimConnect("Managed Client Events", this.Handle, WM_USER_SIMCONNECT, null, 0);

            setButtons(false, true);

            initClientEvent();

        }
        catch (COMException ex)
        {
            displayText("Unable to connect to Prepar3D " + ex.Message);
        }
    }
    else
    {
        displayText("Error - try again");
        closeConnection();

        setButtons(true, false);
    }
}

```

```
private void buttonDisconnect_Click(object sender, EventArgs e)
{
    closeConnection();
    setButtons(true, false);
}

// Response number
int response = 1;

// Output text - display a maximum of 10 lines
string output = "\n\n\n\n\n\n\n\n\n\n";

void displayText(string s)
{
    // remove first string from output
    output = output.Substring(output.IndexOf("\n") + 1);

    // add the new string
    output += "\n" + response++ + ":" + s;

    // display it
    richResponse.Text = output;
}

// End of sample
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

Data Request.cs

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//
// Managed Data Request sample
//
// Click on Connect to try and connect to a running version of Prepar3D
// Click on Request Data any number of times
// Click on Disconnect to close the connection, and then you should
// be able to click on Connect and restart the process
//

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// Add these two statements to all SimConnect clients
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;

namespace Managed_Data_Request
{
    public partial class Form1 : Form
    {

        // User-defined win32 event
        const int WM_USER_SIMCONNECT = 0x0402;

        // SimConnect object
        SimConnect simconnect = null;

        enum DEFINITIONS
        {
            Struct1,
        }

        enum DATA_REQUESTS
        {
            REQUEST_1,
        };

        // this is how you declare a data structure so that
        // simconnect knows how to fill it/read it.
        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
        struct Struct1
        {
            // this is how you declare a fixed size string
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
            public String title;
            public double latitude;
            public double longitude;
            public double altitude;
        };

        public Form1()
        {
            InitializeComponent();
            setButtons(true, false, false);
        }

        // Simconnect client will send a win32 message when there is
        // a packet to process. ReceiveMessage must be called to
        // trigger the events. This model keeps simconnect processing on the main thread.

        protected override void DefWndProc(ref Message m)
        {
            if (m.Msg == WM_USER_SIMCONNECT)
            {
                if (simconnect != null)
                {
                    simconnect.ReceiveMessage();
                }
            }
            else
            {
                base.DefWndProc(ref m);
            }
        }

        private void setButtons(bool bConnect, bool bGet, bool bDisconnect)
        {
            buttonConnect.Enabled = bConnect;
            buttonRequestData.Enabled = bGet;
            buttonDisconnect.Enabled = bDisconnect;
        }

        private void closeConnection()
        {
            if (simconnect != null)
            {
                // Dispose serves the same purpose as SimConnect_Close()
                simconnect.Dispose();
                simconnect = null;
                displayText("Connection closed");
            }
        }

        // Set up all the SimConnect related data definitions and event handlers
        private void initdataRequest()
        {
            try
            {
                // listen to connect and quit msgs
                simconnect.OnRecvOpen += new SimConnect.RecvOpenEventHandler(simconnect_OnRecvOpen);
                simconnect.OnRecvQuit += new SimConnect.RecvQuitEventHandler(simconnect_OnRecvQuit);

                // listen to exceptions
                simconnect.OnRecvException += new SimConnect.RecvExceptionEventHandler(simconnect_OnRecvException);

                // define a data structure
                simconnect.AddToDataDefinition(DEFINITIONS.Struct1, "title", null, SIMCONNECT_DATATYPE.STRING256, 0.0f, SimConnect.SIMCONNECT_UNUSED);
                simconnect.AddToDataDefinition(DEFINITIONS.Struct1, "Plane Latitude", "degrees", SIMCONNECT_DATATYPE.FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
                simconnect.AddToDataDefinition(DEFINITIONS.Struct1, "Plane Longitude", "degrees", SIMCONNECT_DATATYPE.FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
                simconnect.AddToDataDefinition(DEFINITIONS.Struct1, "Plane Altitude", "feet", SIMCONNECT_DATATYPE.FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);

                // IMPORTANT: register it with the simconnect managed wrapper marshaller
                // if you skip this step, you will only receive a uint in the .dwData field.
                simconnect.RegisterDataDefineStruct<Struct1>(DEFINITIONS.Struct1);

                // catch a simobject data request
                simconnect.OnRecvSimobjectDataBytype += new SimConnect.RecvSimobjectDataBytypeEventHandler(simconnect_OnRecvSimobjectDataBytype);
            }
        }
    }
}
```

```

        catch (COMException ex)
        {
            displayText(ex.Message);
        }
    }

void simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
{
    displayText("Connected to Prepar3D");
}

// The case where the user closes Prepar3D
void simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
{
    displayText("Prepar3D has exited");
    closeConnection();
}

void simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    displayText("Exception received: " + data.dwException);
}

// The case where the user closes the client
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    closeConnection();
}

void simconnect_OnRecvSimobjectDataBytype(SimConnect sender, SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE data)
{
    switch ((DATA_REQUESTS)data.dwRequestID)
    {
        case DATA_REQUESTS.REQUEST_1:
            Struct1 s1 = (Struct1)data.dwData[0];
            displayText("title: " + s1.title);
            displayText("Lat: " + s1.latitude);
            displayText("Lon: " + s1.longitude);
            displayText("Alt: " + s1.altitude);
            break;
        default:
            displayText("Unknown request ID: " + data.dwRequestID);
            break;
    }
}

private void buttonConnect_Click(object sender, EventArgs e)
{
    if (simconnect == null)
    {
        try
        {
            // the constructor is similar to SimConnect_Open in the native API
            simconnect = new SimConnect("Managed Data Request", this.Handle, WM_USER_SIMCONNECT, null, 0);

            setButtons(false, true, true);

            initDataRequest();

        }
        catch (COMException ex)
        {
            displayText("Unable to connect to Prepar3D:\n\n" + ex.Message);
        }
    }
    else
    {
        displayText("Error - try again");
        closeConnection();
    }
    setButtons(true, false, false);
}

private void buttonDisconnect_Click(object sender, EventArgs e)
{
    closeConnection();
    setButtons(true, false, false);
}

private void buttonRequestData_Click(object sender, EventArgs e)
{
    // The following call returns identical information to:
    // simconnect.RequestDataOnSimObject(DATA_REQUESTS.REQUEST_1, DEFINITIONS.Struct1, SimConnect.SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD.ONCE);

    simconnect.RequestDataOnSimObjectType(DATA_REQUESTS.REQUEST_1, DEFINITIONS.Struct1, 0, SIMCONNECT_SIMOBJECT_TYPE.USER);
    displayText("Request sent...");
}

// Response number
int response = 1;

// Output text - display a maximum of 10 lines
string output = "\n\n\n\n\n\n\n\n\n\n\n\n";

void displayText(string s)
{
    // remove first string from output
    output = output.Substring(output.IndexOf("\n") + 1);

    // add the new string
    output += "\n" + response++ + ":" + s;

    // display it
    richResponse.Text = output;
}
}

// End of sample

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

Facility Request.cs

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//  

// Managed Facility Request sample  

// Buttons in a dialog are used to requests data on facilities: Airports, Waypoints, NDBs, VORs  

//  

using System;  

using System.Collections.Generic;  

using System.ComponentModel;  

using System.Data;  

using System.Drawing;  

using System.Text;  

using System.Windows.Forms;  

// Add these two statements to all SimConnect clients  

using LockheedMartin.Prepar3D.SimConnect;  

using System.Runtime.InteropServices;  

namespace Managed_Facility_Request  

{  

    public partial class Form1 : Form  

    {  

        // User-defined win32 event  

        const int WM_USER_SIMCONNECT = 0x0402;  

        // SimConnect object  

        SimConnect simconnect = null;  

        enum DATA_REQUESTS  

        {  

            SUBSCRIBE_REQ,  

            NONSUBSCRIBE_REQ,  

        };  

        enum EVENTS  

        {  

            ID0,  

        };  

        public Form1()  

        {  

            InitializeComponent();  

            setButtons(true, false, false);  

        }  

        // Simconnect client will send a win32 message when there is  

        // a packet to process. ReceiveMessage must be called to  

        // trigger the events. This model keeps simconnect processing on the main thread.  

        protected override void DefWndProc(ref Message m)  

        {  

            if (m.Msg == WM_USER_SIMCONNECT)  

            {  

                if (simconnect != null)  

                {  

                    simconnect.ReceiveMessage();  

                }  

            }
            else
            {
                base.DefWndProc(ref m);
            }
        }
  

        private void setButtons(bool bConnect, bool bGet, bool bDisconnect)
        {
            buttonConnect.Enabled = bConnect;
            buttonRequestAirports.Enabled = buttonRequestWaypoints.Enabled
                = buttonRequestNDBs.Enabled
                = buttonRequestVORs.Enabled
                = buttonRequestTacans.Enabled
                = bGet;

            if (bDisconnect)
            {
                checkboxAirportSubscription.Checked = checkboxWaypointsSubscription.Checked
                    = checkboxNDBsSubscription.Checked
                    = checkboxVORsSubscription.Checked
                    = checkboxTacansSubscription.Checked
                    = false;
            }

            labelSubscriptions.Enabled = checkboxAirportSubscription.Enabled
                = checkboxWaypointsSubscription.Enabled
                = checkboxNDBsSubscription.Enabled
                = checkboxVORsSubscription.Enabled
                = checkboxTacansSubscription.Enabled
                = bGet;
            buttonDisconnect.Enabled = bDisconnect;
        }
  

        private void closeConnection()
    }
}
```

```

    {
        if (simconnect != null)
        {
            // Dispose serves the same purpose as SimConnect_Close()
            simconnect.Dispose();
            simconnect = null;
            displayText("Connection closed");
        }
    }

    // Set up all the SimConnect related data definitions and event handlers
    private void initDataRequest()
    {
        try
        {
            // listen to connect and quit msgs
            simconnect.OnRecvOpen += new SimConnect.RecvOpenEventHandler(simconnect_OnRecvOpen);
            simconnect.OnRecvQuit += new SimConnect.RecvQuitEventHandler(simconnect_OnRecvQuit);
            // listen to facilities types
            simconnect.OnRecvAirportList += new SimConnect.RecvAirportListEventHandler(simconnect_OnRecvAirportList);
            simconnect.OnRecvWaypointList += new SimConnect.RecvWaypointListEventHandler(simconnect_OnRecvWaypointList);
            simconnect.OnRecvNdbList += new SimConnect.RecvNdbListEventHandler(simconnect_OnRecvNdbList);
            simconnect.OnRecvVorList += new SimConnect.RecvVorListEventHandler(simconnect_OnRecvVorList);
            simconnect.OnRecvTacanList += new SimConnect.RecvTacanListEventHandler(simconnect_OnRecvTacanList);

            // listen to exceptions
            simconnect.OnRecvException += new SimConnect.RecvExceptionEventHandler(simconnect_OnRecvException);
        }
        catch (COMException ex)
        {
            displayText(ex.Message);
        }
    }

    void Dump(Object item)
    {
        String s = "";
        foreach (System.Reflection.FieldInfo f in item.GetType().GetFields())
        {
            if (!f.FieldType.IsArray)
            {
                s += " " + f.Name + ": " + f.GetValue(item);
            }
        }
        displayText(s);
    }

    void DumpArray(Array rgData)
    {
        foreach (Object item in rgData)
        {
            Dump(item);
        }
    }

    void simconnect_OnRecvAirportList(SimConnect sender, SIMCONNECT_RECV_AIRPORT_LIST data)
    {
        switch ((DATA_REQUESTS)data.dwRequestId)
        {
            case DATA_REQUESTS.SUBSCRIBE_REQ:
            case DATA_REQUESTS.NONSUBSCRIBE_REQ:
                displayText("Airport List:");
                Dump(data);
                DumpArray(data.rgData);
                break;

            default:
                displayText("Unknown request ID: " + data.dwRequestId);
                break;
        }
    }

    void simconnect_OnRecvWaypointList(SimConnect sender, SIMCONNECT_RECV_WAYPOINT_LIST data)
    {
        switch ((DATA_REQUESTS)data.dwRequestId)
        {
            case DATA_REQUESTS.SUBSCRIBE_REQ:
            case DATA_REQUESTS.NONSUBSCRIBE_REQ:
                displayText("Waypoints List:");
                Dump(data);
                DumpArray(data.rgData);
                break;

            default:
                displayText("Unknown request ID: " + data.dwRequestId);
                break;
        }
    }

    void simconnect_OnRecvNdbList(SimConnect sender, SIMCONNECT_RECV_NDB_LIST data)
    {
        switch ((DATA_REQUESTS)data.dwRequestId)
        {
            case DATA_REQUESTS.SUBSCRIBE_REQ:
            case DATA_REQUESTS.NONSUBSCRIBE_REQ:
                displayText("Waypoints List:");
                Dump(data);
                DumpArray(data.rgData);
                break;

            default:
                displayText("Unknown request ID: " + data.dwRequestId);
                break;
        }
    }

    void simconnect_OnRecvVorList(SimConnect sender, SIMCONNECT_RECV_VOR_LIST data)

```

```

    {
        switch ((DATA_REQUESTS)data.dwRequestID)
        {
            case DATA_REQUESTS.SUBSCRIBE_REQ:
            case DATA_REQUESTS.NONSUBSCRIBE_REQ:
                displayText("VOR List:");
                Dump(data);
                DumpArray(data.rgData);
                break;

            default:
                displayText("Unknown request ID: " + data.dwRequestID);
                break;
        }
    }

    void simconnect_OnRecvTacanList(SimConnect sender, SIMCONNECT_RECV_TACAN_LIST data)
    {
        switch ((DATA_REQUESTS)data.dwRequestID)
        {
            case DATA_REQUESTS.SUBSCRIBE_REQ:
            case DATA_REQUESTS.NONSUBSCRIBE_REQ:
                displayText("Tacan List:");
                Dump(data);
                DumpArray(data.rgData);
                break;

            default:
                displayText("Unknown request ID: " + data.dwRequestID);
                break;
        }
    }

    void simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
    {
        displayText("Connected...");
    }

    // The case where the user closes the app
    void simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
    {
        displayText("User has exited");
        closeConnection();
    }

    // The case where the user closes the client
    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        closeConnection();
    }

    void simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
    {
        displayText("Exception received: " + data.dwException);
    }

    private void buttonConnect_Click(object sender, EventArgs e)
    {
        if (simconnect == null)
        {
            try
            {
                // the constructor is similar to SimConnect_Open in the native API
                simconnect = new SimConnect("Managed Facilities Data Request", this.Handle, WM_USER_SIMCONNECT, null, 0);
                initDataRequest();
                setButtons(false, true, true);
            }
            catch (COMException)
            {
                displayText("Unable to connect");
            }
        }
        else
        {
            displayText("Error - try again");
            closeConnection();

            setButtons(true, false, false);
        }
    }

    private void buttonDisconnect_Click(object sender, EventArgs e)
    {
        closeConnection();
        setButtons(true, false, false);
    }

    void displayText(string s)
    {
        richResponse.AppendText(s + "\n");
        richResponse.ScrollToCaret();
    }

    private void buttonRequestAirports_Click(object sender, EventArgs e)
    {
        simconnect.RequestFacilitiesList(SIMCONNECT_FACILITY_LIST_TYPE.AIRPORT, DATA_REQUESTS.NONSUBSCRIBE_REQ);
        displayText("Airports request sent...");
    }

    private void buttonRequestWaypoints_Click(object sender, EventArgs e)
    {
        simconnect.RequestFacilitiesList(SIMCONNECT_FACILITY_LIST_TYPE.WAYPOINT, DATA_REQUESTS.NONSUBSCRIBE_REQ);
        displayText("Waypoints request sent...");
    }
}

```

```

private void buttonRequestNDBs_Click(object sender, EventArgs e)
{
    simconnect.RequestFacilitiesList(SIMCONNECT_FACILITY_LIST_TYPE.NDB, DATA_REQUESTS.NONSUBSCRIBE_REQ);
    displayText("NDBs request sent...");
}

private void buttonRequestVORs_Click(object sender, EventArgs e)
{
    simconnect.RequestFacilitiesList(SIMCONNECT_FACILITY_LIST_TYPE.VOR, DATA_REQUESTS.NONSUBSCRIBE_REQ);
    displayText("VORs request sent...");
}

private void buttonRequestTacans_Click(object sender, EventArgs e)
{
    simconnect.RequestFacilitiesList(SIMCONNECT_FACILITY_LIST_TYPE.TACAN, DATA_REQUESTS.NONSUBSCRIBE_REQ);
    displayText("Tacans request sent...");
}

private void checkboxAirportSubscription_CheckedChanged(object sender, EventArgs e)
{
    if (checkboxAirportSubscription.Enabled)
    {
        if (checkboxAirportSubscription.Checked)
        {
            simconnect.SubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.AIRPORT, DATA_REQUESTS.SUBSCRIBE_REQ);
        }
        else
        {
            simconnect.UnsubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.AIRPORT);
        }
    }
}

private void checkboxWaypointsSubscription_CheckedChanged(object sender, EventArgs e)
{
    if (checkboxWaypointsSubscription.Enabled)
    {
        if (checkboxWaypointsSubscription.Checked)
        {
            simconnect.SubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.WAYPOINT, DATA_REQUESTS.SUBSCRIBE_REQ);
        }
        else
        {
            simconnect.UnsubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.WAYPOINT);
        }
    }
}

private void checkboxNDBsSubscription_CheckedChanged(object sender, EventArgs e)
{
    if (checkboxNDBsSubscription.Enabled)
    {
        if (checkboxNDBsSubscription.Checked)
        {
            simconnect.SubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.NDB, DATA_REQUESTS.SUBSCRIBE_REQ);
        }
        else
        {
            simconnect.UnsubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.NDB);
        }
    }
}

private void checkboxVORsSubscription_CheckedChanged(object sender, EventArgs e)
{
    if (checkboxVORsSubscription.Enabled)
    {
        if (checkboxVORsSubscription.Checked)
        {
            simconnect.SubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.VOR, DATA_REQUESTS.SUBSCRIBE_REQ);
        }
        else
        {
            simconnect.UnsubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.VOR);
        }
    }
}

private void checkboxTacansSubscription_CheckedChanged(object sender, EventArgs e)
{
    if (checkboxTacansSubscription.Enabled)
    {
        if (checkboxTacansSubscription.Checked)
        {
            simconnect.SubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.TACAN, DATA_REQUESTS.SUBSCRIBE_REQ);
        }
        else
        {
            simconnect.UnsubscribeToFacilities(SIMCONNECT_FACILITY_LIST_TYPE.TACAN);
        }
    }
}
}

// End of sample

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

MainWindow.xaml.cs

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//  
//  
// Managed Mission Objects  
//  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Windows;  
using System.Windows.Controls;  
using System.Windows.Data;  
using System.Windows.Documents;  
using System.Windows.Interop;  
using System.Windows.Media;  
using System.Windows.Media.Imaging;  
using System.Windows.Navigation;  
using System.Windows.Shapes;  
  
namespace Managed_Mission_Objects  
{  
    /// <summary>  
    /// Interaction logic for MainWindow.xaml  
    /// </summary>  
    public partial class MainWindow : Window  
    {  
        public MainWindow()  
        {  
            InitializeComponent();  
        }  
  
        public DataModel Model { get; private set; }  
  
        private void Window_Loaded(object sender, RoutedEventArgs e)  
        {  
            DataModel = new DataModel(this, this.RichTextBox);  
  
            // View now has a specific context and can binds to members inside current datamodel  
            this.DataContext = DataModel;  
        }  
  
        private void OnConnectClick(object sender, RoutedEventArgs e)  
        {  
            Model.Connect();  
        }  
  
        private void OnDisconnectClick(object sender, RoutedEventArgs e)  
        {  
            Model.Disconnect();  
        }  
  
        private void OnRequestGoalsClick(object sender, RoutedEventArgs e)  
        {  
            Model.RequestGoals();  
        }  
  
        private void OnRequestMissionObjectivesClick(object sender, RoutedEventArgs e)  
        {  
            Model.RequestMissionObjectives();  
        }  
  
        private void OnRequestFlightSegmentsClick(object sender, RoutedEventArgs e)  
        {  
            Model.RequestFlightSegments();  
        }  
  
        private void Window_Closing(object sender, CancelEventArgs e)  
        {  
            Model.Disconnect();  
        }  
    }  
}
```

DataModel.cs

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.ComponentModel;  
using System.Windows;  
using System.Windows.Interop;  
using System.Windows.Controls;  
  
// Add these two statements to all SimConnect clients  
using LockheedMartin.Prepar3D.SimConnect;  
using System.Runtime.InteropServices;  
  
namespace Managed_Mission_Objects  
{  
    public class DataModel : INotifyPropertyChanged  
    {  
  
        // SimConnect object  
        public SimConnect SimConnectModule { get; private set; }  
  
        public RichTextBox OutputLog { get; private set; }  
  
        public IntPtr Handle { get; private set; }  
  
        public bool IsConnected  
        {  
            get { return SimConnectModule != null; }  
        }  
  
        public string SimConnectStatus  
        { get; private set; }  
  
        public HwndSource HandleSource  
        { get; private set; }  
  
        private const int WM_USER_SIMCONNECT = 0x0402;  
  
        private const string DEFAULT_SIMCONNECT_STATUS = "";  
  
        enum DATA_REQUESTS  
        {  
            REQUEST_COUNT,  
            REQUEST_INDEX,  
            REQUEST_GUID,  
        };  
  
        enum EVENT_ID  
        {  
            FLIGHT_SEGMENT_READY_FOR_GRADING,  
        };  
  
        public DataModel(Window window, RichTextBox richTextBox)  
        {  
            this.SimConnectModule = null;  
  
            // Create an event handle for the WPF window to listen for SimConnect events
```

```

Handle = new WindowInteropHelper(window).Handle; // Get handle of main WPF Window
HandleSource = HwndSource.FromHwnd(Handle); // Get source of handle in order to add event handlers to it
HandleSource.AddHook(HandleSimConnectEvents);

OutputLog = richTextBox;
SimConnectStatus = DEFAULT_SIMCONNECT_STATUS;
}

~DataModel()
{
    if (HandleSource != null)
    {
        HandleSource.RemoveHook(HandleSimConnectEvents);
    }
}

private void WriteToOutput(string outputString)
{
    OutputLog.AppendText("\n" + outputString);
    OutputLog.ScrollToEnd();
}

public void Connect()
{
    if (SimConnectModule == null)
    {
        try
        {
            // the constructor is similar to SimConnect_Open in the native API
            SimConnectModule = new SimConnect("Managed Mission Object", this.Handle, WM_USER_SIMCONNECT, null, 0);

            OnPropertyChanged("IsConnected");

            SetupEventHandlers();
        }
        catch (COMException ex)
        {
            WriteToOutput("Unable to connect to Prepar3D:\n\n" + ex.Message);
        }
    }
    else
    {
        WriteToOutput("Error - try again");
        Disconnect();
    }
}

private void SetupEventHandlers()
{
    try
    {
        // listen to connect and quit msgs
        SimConnectModule.OnRecvOpen += new SimConnect.RecvOpenEventHandler(Simconnect_OnRecvOpen);
        SimConnectModule.OnRecvQuit += new SimConnect.RecvQuitEventHandler(Simconnect_OnRecvQuit);

        // listen to exceptions
        SimConnectModule.OnRecvException += new SimConnect.RecvExceptionEventHandler(Simconnect_OnRecvException);

        SimConnectModule.OnRecvGoal += new SimConnect.RecvGoalEventHandler(Simconnect_OnRecvGoal);

        SimConnectModule.OnRecvMissionObjectCount += new SimConnect.RecvMissionObjectCountEventHandler(SimConnect_OnRecvMissionObjectCount);

        SimConnectModule.OnRecvMissionObjective += new SimConnect.RecvMissionObjectiveEventHandler(Simconnect_OnRecvMissionObjective);

        SimConnectModule.OnRecvFlightSegment += new SimConnect.RecvFlightSegmentEventHandler(Simconnect_OnRecvFlightSegment);

        SimConnectModule.OnRecvParameterRange += new SimConnect.RecvParameterRangeEventHandler(Simconnect_OnRecvParameterRange);

        SimConnectModule.OnRecvFlightSegmentReadyForGrading += new SimConnect.RecvFlightSegmentReadyForGradingEventHandler(Simconnect_OnRecvFlightSegmentReadyForGrading);

        SimConnectModule.SubscribeToSystemEvent(EVENT_ID.FLIGHT_SEGMENT_READY_FOR_GRADING, "FlightSegmentReadyForGrading");
        SimConnectModule.SetSystemEventState(EVENT_ID.FLIGHT_SEGMENT_READY_FOR_GRADING, SIMCONNECT_STATE.ON);
    }
    catch (COMException ex)
    {
        WriteToOutput(ex.Message);
    }
}

public void Disconnect()
{
    if (SimConnectModule != null)
    {
        SimConnectModule.UnsubscribeFromSystemEvent(EVENT_ID.FLIGHT_SEGMENT_READY_FOR_GRADING);
        // Dispose serves the same purpose as SimConnect_Close()
        SimConnectModule.Dispose();
        SimConnectModule = null;
        OnPropertyChanged("IsConnected");

        SimConnectStatus = "Disconnected from Prepar3D";
        OnPropertyChanged("SimConnectStatus");

        WriteToOutput("Connection closed");
    }
}

void Simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
{
    WriteToOutput("Connected to Prepar3D");
    SimConnectStatus = "Connected to Prepar3D";
    OnPropertyChanged("SimConnectStatus");
}

// The case where the user closes Prepar3D
void Simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
{
    WriteToOutput("Prepar3D has exited");
    Disconnect();
}

void Simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    WriteToOutput("Exception received: " + data.dwException);
}

public void Simconnect_OnRecvGoal(SimConnect sender, SIMCONNECT_RECV_GOAL data)
{
    if ((DATA_REQUESTS)data.dwRequestID == DATA_REQUESTS.REQUEST_GUID)
    {
        WriteToOutput("Order: " + data.dwOrder);
        WriteToOutput("Point Value: " + data.dwPointValue);
        WriteToOutput("Guid: " + data.guidInstanceId);
        WriteToOutput("IsOptional: " + (data.isOptional != 0));
        WriteToOutput("Text: " + data.szGoalText);

        string goalState;
        switch (data.eGoalState)
        {
            case SIMCONNECT_GOAL_STATE.GOAL_COMPLETED:
                goalState = "Completed";
                break;
            case SIMCONNECT_GOAL_STATE.GOAL_FAILED:
                goalState = "Failed";
                break;
            case SIMCONNECT_GOAL_STATE.GOAL_PENDING:
                goalState = "Pending";
                break;
            default:
                goalState = "Invalid goal state";
                break;
        }

        WriteToOutput("State: " + goalState);
    }
    else
    {
}
}

```

```

        SimConnectModule.RequestGoalDataByGUID(DATA_REQUESTS.REQUEST_GUID, data.guidInstanceID);
        WriteToOutput("Goal Request By GUID");
    }

}

public void Simconnect_OnRecvMissionObjective(SimConnect sender, SIMCONNECT_RECV_MISSION_OBJECTIVE data)
{
    if ((DATA_REQUESTS)data.dwRequestId == DATA_REQUESTS.REQUEST_GUID)
    {
        WriteToOutput("Order: " + data.dwOrder);
        WriteToOutput("Guid: " + data.guidInstanceID);
        WriteToOutput("IsOptional: " + (data.bOptional != 0));
        WriteToOutput("Text: " + data.szMissionObjectiveText);
        WriteToOutput("Point Value: " + data.dwPointValue);
        WriteToOutput("Current Score: " + data.dwCurrentScore);
        WriteToOutput("Pass Value: " + data.dwPassValue);
        WriteToOutput("Total Possible: " + data.dwTotalPossiblePoints);
        WriteToOutput("Passed: " + (data.isObjectivePassed != 0));

        string missionObjectiveState;
        switch (data.eMissionObjectiveStatus)
        {
            case SIMCONNECT_MISSION_OBJECTIVE_STATUS.PASSED:
                missionObjectiveState = "Passed";
                break;
            case SIMCONNECT_MISSION_OBJECTIVE_STATUS.FAILED:
                missionObjectiveState = "Failed";
                break;
            case SIMCONNECT_MISSION_OBJECTIVE_STATUS.PENDING:
                missionObjectiveState = "Pending";
                break;
            default:
                missionObjectiveState = "Invalid goal state";
                break;
        }
        WriteToOutput("Status: " + missionObjectiveState);
    }
    else
    {
        SimConnectModule.RequestMissionObjectiveDataByGUID(DATA_REQUESTS.REQUEST_GUID, data.guidInstanceID);
        WriteToOutput("Mission Objective Request By GUID");
    }
}

void SimConnect_OnRecvMissionObjectCount(SimConnect sender, SIMCONNECT_RECV_MISSION_OBJECT_COUNT data)
{
    if (data.eMissionObjectType == SIMCONNECT_MISSION_OBJECT_TYPE.GOAL)
    {
        for (int i = 0; i < data.dwCount; ++i)
        {
            SimConnectModule.RequestGoalDataByIndex(DATA_REQUESTS.REQUEST_INDEX, i);
            WriteToOutput("Goal Request Sent");
        }
    }
    else if (data.eMissionObjectType == SIMCONNECT_MISSION_OBJECT_TYPE.MISSION_OBJECTIVE)
    {
        for (int i = 0; i < data.dwCount; ++i)
        {
            SimConnectModule.RequestMissionObjectiveDataByIndex(DATA_REQUESTS.REQUEST_INDEX, i);
            WriteToOutput("Mission Objective Request Sent");
        }
    }
    else if (data.eMissionObjectType == SIMCONNECT_MISSION_OBJECT_TYPE.FLIGHT_SEGMENT)
    {
        for (int i = 0; i < data.dwCount; ++i)
        {
            SimConnectModule.RequestFlightSegmentDataByIndex(DATA_REQUESTS.REQUEST_INDEX, i);
            WriteToOutput("Flight Segment Request Sent");
        }
    }
}

void Simconnect_OnRecvFlightSegment(SimConnect sender, SIMCONNECT_RECV_FLIGHT_SEGMENT data)
{
    WriteToOutput("Parameter Count: " + data.dwParameterCount);
    WriteToOutput("Total Range Count: " + data.dwTotalRangeCount);
    WriteToOutput("Guid: " + data.guidInstanceID);
    WriteToOutput("Goal Guid: " + data.guidSegmentGoalID);

    for (int i = 0; i < data.dwTotalRangeCount; ++i)
    {
        SimConnectModule.RequestFlightSegmentRangeData(DATA_REQUESTS.REQUEST_INDEX, data.guidInstanceID, i);
        WriteToOutput("Range Request Sent");
    }

    if ((DATA_REQUESTS)data.dwRequestId == DATA_REQUESTS.REQUEST_GUID)
    {
        if (data.guidSegmentGoalID != Guid.Empty) //If the flight segment has a goal, pass it
        {
            SimConnectModule.ResolveGoal(data.guidSegmentGoalID, SIMCONNECT_GOAL_RESOLUTION.COMPLETED);
        }
    }
}

void Simconnect_OnRecvParameterRange(SimConnect sender, SIMCONNECT_RECV_PARAMETER_RANGE data)
{
    WriteToOutput("Parameter: " + data.szParameterName);
    WriteToOutput("Range: " + data.szRangeName);
    WriteToOutput("MaxOverMeasured: " + data.dwMaxOverMeasured);
    WriteToOutput("MinUnderMeasured: " + data.dwMinUnderMeasured);
    WriteToOutput("Exceeded Count: " + data.dwExceededCount);
}

void Simconnect_OnRecvFlightSegmentReadyForGrading(SimConnect sender, SIMCONNECT_RECV_FLIGHT_SEGMENT_READY_FOR_GRADING recEvent)
{
    switch (recEvent.uEventID)
    {
        case (uint)EVENT_ID.FLIGHT_SEGMENT_READY_FOR_GRADING:
            WriteToOutput("Flight Segment with GUID " + recEvent.guidInstanceID + " Ready For Grading");
            SimConnectModule.RequestFlightSegmentDataByGUID(DATA_REQUESTS.REQUEST_GUID, recEvent.guidInstanceID);
            break;
        default:
            WriteToOutput("Unknown event");
            break;
    }
}

public void RequestGoals()
{
    SimConnectModule.RequestGoalCount(DATA_REQUESTS.REQUEST_COUNT);
}

public void RequestMissionObjectives()
{
    SimConnectModule.RequestMissionObjectiveCount(DATA_REQUESTS.REQUEST_COUNT);
}

public void RequestFlightSegments()
{
    SimConnectModule.RequestFlightSegmentCount(DATA_REQUESTS.REQUEST_COUNT);
}

private IntPtr HandleSimConnectEvents(IntPtr hWnd, int message, IntPtr wParam, IntPtr lParam, ref bool isHandled)
{
    isHandled = false;

    switch (message)
    {
        case WM_USER_SIMCONNECT:
            if (SimConnectModule != null)

```

```
        {
            try
            {
                this.SimConnectModule.ReceiveMessage();
            }
            catch { RecoverFromError(); }

            isHandled = true;
        }
        break;
    default:
        break;
}
return IntPtr.Zero;
}

private void RecoverFromError()
{
    Disconnect();
    Connect();
}

#region INotifyPropertyChanged : Allows properties that are bound to tell their controls that they're bound to that they've been updated
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
#endregion
}
```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

```
ObserverControl.cs
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
///
/// Managed Observer Control Sample
///
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// Add these two statements to all SimConnect clients
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;

/**+
 * The focus type used for observer focal-length and focal-target calculations.
 * This either causes the focus to be auto-calculated to be place on the terrain surface whenever possible, or to fix the focus within relative observer-space.
 */
enum OBSERVER_FOCUS_TYPE
{
    OBSERVER_FOCUS_WORLD,    /**< The observer focal point is automatically calculated during movement to intersect with the simulation world. */
    OBSERVER_FOCUS_LOCAL     /**< The observer focal point is locked relative to the camera position. Moving the camera will move the focal point, but will not change the focal distance. */
};

/**+
 * The regime that the observer is allowed to operate within.
 */
enum OBSERVER_REGIME_TYPE
{
    OBSERVER_TELLURIAN,      /**< The observer is limited by the earth only; that is, the position is restricted from going below the sea-floor or below the ground, but may pass through water surfaces. */
    OBSERVER_TERRESTRIAL,    /**< The observer is limited to above-water movement; that is, the position is restricted from going below the ground or below any water surface. */
    OBSERVER_GHOST           /**< The observer is not limited by any terrain features; position may freely pass through any object or terrain feature. */
};

namespace ObserverControl
{
    public partial class ObserverControl : Form
    {
        // User-defined win32 event
        const int WM_USER_SIMCONNECT = 0x0402;

        // SimConnect object
        SimConnect simconnect = null;

        enum DATA_REQUESTS
        {
            REQUEST_1,
        };

        enum NOTIFICATION_GROUPS
        {
            GROUP0,
        };

        public ObserverControl()
        {
            InitializeComponent();
            setButtons(true, false);
        }

        // Simconnect client will send a win32 message when there is
        // a packet to process. ReceiveMessage must be called to
        // trigger the events. This model keeps simconnect processing on the main thread.
        protected override void DefWndProc(ref Message m)
        {
            if (m.Msg == WM_USER_SIMCONNECT)
            {
                if (simconnect != null)
                {
                    simconnect.ReceiveMessage();
                }
            }
            else
            {
                base.DefWndProc(ref m);
            }
        }

        private void setButtons(bool bConnect, bool bDisconnect)
        {
            buttonConnect.Enabled = bConnect;
            buttonDisconnect.Enabled = bDisconnect;
            buttonCreateObserver.Enabled = bDisconnect;

            buttonSetPosition.Enabled = bDisconnect;
            buttonSetElevation.Enabled = bDisconnect;
            buttonSetPitchYView.Enabled = bDisconnect;
            buttonSetStepSize.Enabled = bDisconnect;
            buttonSetFocus.Enabled = bDisconnect;
            buttonSetRegime.Enabled = bDisconnect;
        }

        private void closeConnection()
        {
            if (simconnect != null)
            {
                // Dispose serves the same purpose as SimConnect_Close()
                simconnect.Dispose();
                simconnect = null;
                displayText("Connection closed");
            }
        }

        // Set up all the SimConnect related event handlers
        private void initClientEvent()
        {
            try
            {
                // listen to connect and quit msgs
                simconnect.OnRecvOpen += new SimConnect.RecvOpenEventHandler(simconnect_OnRecvOpen);
                simconnect.OnRecvQuit += new SimConnect.RecvQuitEventHandler(simconnect_OnRecvQuit);

                // listen to exceptions
                simconnect.OnRecvException += new SimConnect.RecvExceptionEventHandler(simconnect_OnRecvException);

                // catch a simobject data request
                simconnect.OnRecvObserverData += new SimConnect.RecvObserverDataEventHandler(simconnect_OnRecvObserverData);
            }
            catch (COMException ex)
            {
                displayText(ex.Message);
            }
        }

        void simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
        {
            displayText("Connected to Prepar3D");
        }

        // The case where the user closes Prepar3D
        void simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
        {
            displayText("Prepar3D has exited");
            closeConnection();
        }

        void simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
        {
```

```

        displayText("Exception received: " + data.dwException);
    }

    // The case where the user closes the client
    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        closeConnection();
    }

    // After calling RequestObserverData, Prepar3D sends back SIMCONNECT_RECV_OBSERVER_DATA
    void simconnect_OnRecvObserverData(SimConnect sender, SIMCONNECT_RECV_OBSERVER_DATA data)
    {
        switch ((DATA_REQUESTS)data.dwRequestID)
        {
            case DATA_REQUESTS.REQUEST_1:
                displayText("received Data for " + data.szObserverName);

                textBoxXPosLat.Text = data.ObserverData.Position.Latitude.ToString();
                textBoxXPosLong.Text = data.ObserverData.Position.Longitude.ToString();
                textBoxXPosAlt.Text = data.ObserverData.Position.Altitude.ToString();

                textBoxHeading.Text = data.ObserverData.Rotation Heading.ToString();
                textBoxPitch.Text = data.ObserverData.Rotation.Pitch.ToString();
                textBoxBank.Text = data.ObserverData.Rotation.Bank.ToString();

                textBoxFOVH.Text = data.ObserverData.FieldOfViewH.ToString();
                textBoxFOVV.Text = data.ObserverData.FieldOfViewV.ToString();

                textBoxMSSLinear.Text = data.ObserverData.LinearStep.ToString();
                textBoxSSAngular.Text = data.ObserverData.AngularStep.ToString();

                textBoxMFocalLength.Text = data.ObserverData.FocalLength.ToString();

                // Focus
                if (data.ObserverData.FocusFixed == (int)OBSERVER_FOCUS_TYPE.OBSERVER_FOCUS_LOCAL)
                {
                    radioButtonMFocusPoint.Checked = true;
                }
                else
                {
                    radioButtonMFocusWorld.Checked = true;
                }

                // Regime
                if (data.ObserverData.Regime == (int)OBSERVER_REGIME_TYPE.OBSERVER_GHOST)
                {
                    checkBoxMGhostMode.Checked = true;
                    checkBoxMPassWatter.Checked = true;
                }
                else if (data.ObserverData.Regime == (int)OBSERVER_REGIME_TYPE.OBSERVER_TELLURIAN)
                {
                    checkBoxMGhostMode.Checked = false;
                    checkBoxMPassWatter.Checked = true;
                }
                else
                {
                    checkBoxMGhostMode.Checked = false;
                    checkBoxMPassWatter.Checked = false;
                }

                break;

            default:
                displayText("Unknown request ID: " + data.dwRequestID);
                break;
        }
    }

    private void buttonConnect_Click(object sender, EventArgs e)
    {
        if (simconnect == null)
        {
            try
            {
                // the constructor is similar to SimConnect_Open in the native API
                simconnect = new SimConnect("Managed ObserverControl", this.Handle, WM_USER_SIMCONNECT, null, 0);

                setButtons(false, true);

                initClientEvent();
            }
            catch (COMException ex)
            {
                displayText("Unable to connect to Prepar3D " + ex.Message);
            }
        }
        else
        {
            displayText("Error - try again");
            closeConnection();
        }

        setButtons(true, false);
    }

    private void buttonDisconnect_Click(object sender, EventArgs e)
    {
        closeConnection();
        setButtons(true, false);
    }

    // Response number
    int response = 1;

    // Output text - display a maximum of 5 lines
    string output = "\n\n\n\n\n";

    void displayText(string s)
    {
        // remove first string from output
        output = output.Substring(output.IndexOf("\n") + 1);

        // add the new string
        output += "\n" + response++ + ":" + s;

        // display it
        richResponse.Text = output;
    }

    // Create Observer Calls -----
    // Uses all data entered in the create tab to create the named observer
    unsafe private void buttonCreateObserver_Click(object sender, EventArgs e)
    {
        SIMCONNECT_DATA_OBSERVER observerData = new SIMCONNECT_DATA_OBSERVER();

        observerData.Position.Latitude = double.Parse(textBoxCPosLat.Text);
        observerData.Position.Longitude = double.Parse(textBoxCPosLong.Text);
        observerData.Position.Altitude = double.Parse(textBoxCPosAlt.Text);

        observerData.Rotation.Heading = double.Parse(textBoxCHeading.Text);
        observerData.Rotation.Pitch = double.Parse(textBoxCPitch.Text);
        observerData.Rotation.Bank = double.Parse(textBoxCBank.Text);

        observerData.FieldOfViewH = float.Parse(textBoxCFOVH.Text);
        observerData.FieldOfViewV = float.Parse(textBoxCFOVV.Text);

        observerData.LinearStep = float.Parse(textBoxCSSLinear.Text);
        observerData.AngularStep = float.Parse(textBoxCSSAngular.Text);

        observerData.FocalLength = float.Parse(textBoxCFocalLength.Text);

        if (radioButtonFocusPoint.Checked)
        {
            observerData.FocusFixed = (int)OBSERVER_FOCUS_TYPE.OBSERVER_FOCUS_LOCAL;
        }
        else
        {
            observerData.FocusFixed = (int)OBSERVER_FOCUS_TYPE.OBSERVER_FOCUS_WORLD;
        }

        if (checkBoxCGhostMode.Checked)
        {
            observerData.Regime = (int)OBSERVER_REGIME_TYPE.OBSERVER_GHOST;
        }
        else if (checkBoxCPassWatter.Checked)
        {
            observerData.Regime = (int)OBSERVER_REGIME_TYPE.OBSERVER_TELLURIAN;
        }
    }
}

```

```

        }
    else
    {
        observerData.Regime = (int)OBSERVER_REGIME_TYPE.OBSERVER_TERRESTRIAL;
    }

    simconnect.CreateObserver(textBoxCObserverName.Text, observerData);
    displayText("Observer \\" + textBoxCObserverName.Text + "\\ created");
}

private void checkBoxCGhostMode_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxCGhostMode.Checked)
    {
        checkBoxCPassWater.Checked = true;
        checkBoxCPassWater.Enabled = false;
    }
    else
    {
        checkBoxCPassWater.Enabled = true;
    }
}

// Modify Observer Calls -----
private void buttonPopulateFields_Click(object sender, EventArgs e)
{
    simconnect.RequestObserverData(DATA_REQUESTS.REQUEST_1, textBoxMObserverName.Text);
}

private void buttonSetPosition_Click(object sender, EventArgs e)
{
    SIMCONNECT_DATA_LATLONALT observerLLA = new SIMCONNECT_DATA_LATLONALT();

    observerLLA.Latitude = double.Parse(textBoxMPosLat.Text);
    observerLLA.Longitude = double.Parse(textBoxMPosLong.Text);
    observerLLA.Altitude = double.Parse(textBoxMPosAlt.Text);

    simconnect.SetObserverPosition(textBoxMObserverName.Text, observerLLA);
}

private void buttonSetRotation_Click(object sender, EventArgs e)
{
    SIMCONNECT_DATA_PBH observerPBH = new SIMCONNECT_DATA_PBH();

    observerPBH.Pitch = double.Parse(textBoxMPitch.Text);
    observerPBH.Bank = double.Parse(textBoxMBank.Text);
    observerPBH.Heading = double.Parse(textBoxMHeading.Text);

    simconnect.SetObserverRotation(textBoxMObserverName.Text, observerPBH);
}

private void buttonSetFieldOfView_Click(object sender, EventArgs e)
{
    simconnect.SetObserverFieldOfView(textBoxMObserverName.Text, float.Parse(textBoxMFOVH.Text), float.Parse(textBoxMFOVV.Text));
}

private void buttonSetStepSize_Click(object sender, EventArgs e)
{
    simconnect.SetObserverStepSize(textBoxMObserverName.Text, float.Parse(textBoxMSSLinear.Text), float.Parse(textBoxMSSAngular.Text));
}

private void buttonSetFocus_Click(object sender, EventArgs e)
{
    simconnect.SetObserverFocusFixed(textBoxMObserverName.Text, radioButtonMFocusPoint.Checked);
    simconnect.SetObserverFocalLength(textBoxMObserverName.Text, float.Parse(textBoxMFocalLength.Text));
}

private void buttonSetRegime_Click(object sender, EventArgs e)
{
    if (checkBoxMGhostMode.Checked)
    {
        simconnect.SetObserverRegime(textBoxMObserverName.Text, (int)OBSERVER_REGIME_TYPE.OBSERVER_GHOST);
    }
    else if (checkBoxMPassWater.Checked)
    {
        simconnect.SetObserverRegime(textBoxMObserverName.Text, (int)OBSERVER_REGIME_TYPE.OBSERVER_TELLURIAN);
    }
    else
    {
        simconnect.SetObserverRegime(textBoxMObserverName.Text, (int)OBSERVER_REGIME_TYPE.OBSERVER_TERRESTRIAL);
    }
}

private void checkBoxMGhostMode_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxMGhostMode.Checked)
    {
        checkBoxMPassWater.Checked = true;
        checkBoxMPassWater.Enabled = false;
    }
    else
    {
        checkBoxMPassWater.Enabled = true;
    }
}
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

System Event.cs

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
//  
//  
// Managed System Event sample  
//  
// Click on the buttons to receive 4 second and Sim start and stop notifications  
  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Text;  
using System.Windows.Forms;  
  
// Add these two statements to all SimConnect clients  
using LockheedMartin.Prepar3D.SimConnect;  
using System.Runtime.InteropServices;  
  
namespace Managed_System_Event  
{  
    public partial class Form1 : Form  
    {  
        // User-defined win32 event  
        const int WM_USER_SIMCONNECT = 0x0402;  
  
        // SimConnect object  
        SimConnect simconnect = null;  
  
        // System state switches  
        int P3D_System_4      = 0;  
        int P3D_System_Sim   = 0;  
  
        enum REQUESTS  
        {  
            REQUEST_4S,  
            REQUEST_SIMSTATE,  
        };  
  
        enum EVENTS  
        {  
            SIMSTART,  
            SIMSTOP,  
            FOURSECS,  
        };  
  
        public Form1()  
        {  
            InitializeComponent();  
  
            setButtons(true, false, false);  
        }  
        // Simconnect client will send a win32 message when there is  
        // a packet to process. ReceiveMessage must be called to  
        // trigger the events. This model keeps simconnect processing on the main thread.  
  
        protected override void DefWndProc(ref Message m)  
        {  
            if (m.Msg == WM_USER_SIMCONNECT)  
            {  
                if (simconnect != null)  
                {  
                    simconnect.ReceiveMessage();  
                }  
            }  
            else  
            {  
                base.DefWndProc(ref m);  
            }  
        }  
  
        private void setButtons(bool bConnect, bool bGet, bool bDisconnect)  
        {  
            buttonConnect.Enabled = bConnect;  
            buttonRequest4.Enabled = bGet;  
            buttonSimStart.Enabled = bGet;  
            buttonDisconnect.Enabled = bDisconnect;  
        }  
  
        private void closeConnection()  
        {  
            if (simconnect != null)  
            {  
                simconnect.Close();  
            }  
        }  
    }  
}
```

```

        // Unsubscribe from all the system events
        simconnect.UnsubscribeFromSystemEvent(EVENTS.FOURSECS);
        simconnect.UnsubscribeFromSystemEvent(EVENTS.SIMSTART);
        simconnect.UnsubscribeFromSystemEvent(EVENTS.SIMSTOP);

        // Dispose serves the same purpose as SimConnect_Close()
        simconnect.Dispose();
        simconnect = null;
        displayText("Connection closed");
    }
}

// Set up all the SimConnect related event handlers
private void initSystemEvent()
{
    try
    {
        // listen to connect and quit msgs
        simconnect.OnRecvOpen += new SimConnect.RecvOpenEventHandler(simconnect_OnRecvOpen);
        simconnect.OnRecvQuit += new SimConnect.RecvQuitEventHandler(simconnect_OnRecvQuit);

        // listen to exceptions
        simconnect.OnRecvException += new SimConnect.RecvExceptionHandler(simconnect_OnRecvException);

        // listen to events
        simconnect.OnRecvEvent += new SimConnect.RecvEventEventHandler(simconnect_OnRecvEvent);

        // Subscribe to system events
        simconnect.SubscribeToSystemEvent(EVENTS.FOURSECS, "4sec");
        simconnect.SubscribeToSystemEvent(EVENTS.SIMSTART, "SimStart");
        simconnect.SubscribeToSystemEvent(EVENTS.SIMSTOP, "SimStop");

        // Initially turn the events off
        simconnect.SetSystemEventState(EVENTS.FOURSECS, SIMCONNECT_STATE.OFF);
        simconnect.SetSystemEventState(EVENTS.SIMSTART, SIMCONNECT_STATE.OFF);
        simconnect.SetSystemEventState(EVENTS.SIMSTOP, SIMCONNECT_STATE.OFF);
    }
    catch (COMException ex)
    {
        displayText(ex.Message);
    }
}

void simconnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
{
    displayText("Connected to Prepar3D");
}

// The case where the user closes Prepar3D
void simconnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
{
    displayText("Prepar3D has exited");
    closeConnection();
}

// The case where the user closes the client
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    buttonDisconnect_Click(sender, null);
}

void simconnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    displayText("Exception received: " + data.dwException);
}

void simconnect_OnRecvEvent(SimConnect sender, SIMCONNECT_RECV_EVENT recEvent)
{
    switch (recEvent.uEventID)
    {
        case (uint) EVENTS.SIMSTART:

            displayText("Sim running");
            break;

        case (uint) EVENTS.SIMSTOP:

            displayText("Sim stopped");
            break;

        case (uint) EVENTS.FOURSECS:

            displayText("4s tick");
            break;
    }
}

private void buttonConnect_Click(object sender, EventArgs e)
{
    if (simconnect == null)
    {
        try
        {
            // the constructor is similar to SimConnect_Open in the native API
            simconnect = new SimConnect("Managed System Event", this.Handle, WM_USER_SIMCONNECT, null, 0);

            setButtons(false, true, true);
        }
    }
}

```


PREPAR3D

```
View.xaml.cs
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
///
/// Managed Weapon Station Selection
///

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.ComponentModel;

namespace Managed_Weapon_Station_Selection
{
    /// <summary>
    /// Interaction logic for View.xaml
    /// </summary>
    public partial class View : Window
    {

        public View()
        {
            InitializeComponent();
        }

        public DataModel DataModel
        { get; private set; }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            this.DataModel = new DataModel(this, StationsDataGrid);

            // View now has a specific context and can binds to members inside current datamodel
            this.DataContext = DataModel;
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            // Isn't connected, try to connect
            if (!this.DataModel.IsConnected)
            {
                string errorMessage;
                bool wasSuccess = this.DataModel.Connect(out errorMessage);

                // Start monitoring the user's SimObject. This will continuously monitor information
                // about the user's Stations attached to their SimObject.
                if (wasSuccess)
                {
                    this.DataModel.StartMonitoring();
                }
                // Show error to user
                else
                {
                    MessageBox.Show(errorMessage);
                }
            }
            // Is connected, try to disconnect
            else
            {
                this.DataModel.Disconnect();
            }
        }

        private void Window_Closing(object sender, CancelEventArgs e)
        {
            if (this.DataModel.IsConnected)
            {
                this.DataModel.Disconnect();
            }
        }

    }
}
```

```
DataModel.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

// Add these two statements to all SimConnect clients
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;

using System.Windows.Interop;
using System.ComponentModel;
using System.Windows.Controls;
using System.Collections.ObjectModel;

namespace Managed_Weapon_Station_Selection
{
    public class DataModel : INotifyPropertyChanged
    {
        public DataModel(Window window, ItemsControl itemsControl)
        {
            this.SimConnect = null;

            // Create an event handle for the WPF window to listen for SimConnect events
            Handle = new WindowInteropHelper(window).Handle; // Get handle of main WPF Window
            HandleSource = HwndSource.FromHwnd(Handle); // Get source of handle in order to add event handlers to it
            HandleSource.AddHook(HandleSimConnectEvents);

            // Set default values for properties
            this.NumberOfStations = DEFAULT_NUMBER_OF_STATIONS;
            this.NumberOfStationsProcessed = DEFAULT_NUMBER_OF_STATIONS_PROCESSED;
            this.Title = DEFAULT_TITLE;
            this.SimConnectStatus = DEFAULT_SIMCONNECT_STATUS;
            this.Stations = new ObservableCollection<Station>();

            // Set binding source
            itemsControl.ItemsSource = this.Stations;
        }

        ~DataModel()
        {
            if (HandleSource != null)
            {
                HandleSource.RemoveHook(HandleSimConnectEvents);
            }
        }

        // User-defined win32 event
        private const int WM_USER_SIMCONNECT = 0x0402;

        private const int DEFAULT_NUMBER_OF_STATIONS = 0;
        private const int DEFAULT_NUMBER_OF_STATIONS_PROCESSED = 0;
```

```

private const string DEFAULT_TITLE = "[None]";
private const string DEFAULT_SIMCONNECT_STATUS = "";

// SimConnect object
public SimConnect SimConnect
{ get; private set; }

public bool IsConnected
{
    get { return SimConnect != null; }
}

public IntPtr Handle
{ get; private set; }

public HwndSource HandleSource
{ get; private set; }

public enum DataIdentifier
{
    RecieveSimObjectGeneral = 0,
    RecieveStationIndex = 1,
    RecieveStationEnd = 1000,
    SendStationAtIndex = 1001,
    SendStationEnd = 2000
}

public enum EventIdentifier
{
    Every6Hz = 0
}

// Data Structure associated with STATIONS_GENERAL
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
struct SimObjectGeneral
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public String Title;
    public double NumberOfStations;
}

// Data Structure associated with STATION_AT_INDEX
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
struct StationAtIndex
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public String Title;
    public double IsSelected;
}

public string SimConnectStatus
{ get; private set; }

public string Title
{ get; private set; }

public int NumberOfStations
{ get; private set; }

private int NumberOfStationsProcessed
{ get; set; }

public ObservableCollection<Station> Stations
{ get; set; }

#region Connection Methods

public bool Connect(out string errorMessage)
{
    if (this.SimConnect == null)
    {
        try
        {
            if (Handle == null)
            {
                errorMessage = "Do not have access to Window Handle";
                return false;
            }

            // the constructor is similar to SimConnect_Open in the native API
            this.SimConnect = new SimConnect("Managed Data Request", Handle, WM_USER_SIMCONNECT, null, 0);

            OnPropertyChanged("IsConnected");

            // Subscribe to events that we care about. We would like to know when we finish connecting, when the application exits,
            // when an exception is raised, when events happen, and when simobject data is returned (respectively)
            this.SimConnect.OnRecvOpen += new LockheedMartin.Prepar3D.SimConnect.RecvEventEventHandler(SimConnect_OnRecvOpen);
            this.SimConnect.OnError += new LockheedMartin.Prepar3D.SimConnect.ErrorEventEventHandler(SimConnect_OnError);
            this.SimConnect.OnRecvException += new LockheedMartin.Prepar3D.SimConnect.SimConnect.RecvExceptionEventHandler(SimConnect_OnRecvException);
            this.SimConnect.OnRecvEvent += new LockheedMartin.Prepar3D.SimConnect.SimConnect.RecvEventEventHandler(SimConnect_OnRecvEvent);
            this.SimConnect.OnRecvSimobjectDataByType += new LockheedMartin.Prepar3D.SimConnect.RecvSimobjectDataByTypeEventHandler(SimConnect_OnRecvSimobjectDataBytype);

            // Subscribe to system events
            this.SimConnect.SubscribeToSystemEvent(EventIdentifier.Every6Hz, "1sec"); //"/"6Hz";

            // Initially turn the events off
            this.SimConnect.SetSystemEventState(EventIdentifier.Every6Hz, SIMCONNECT_STATE.OFF);

            // Successfully connected
            errorMessage = string.Empty;
            return true;
        }
        catch (COMException ex)
        {
            // Something went wrong, Main window can display it to user.
            errorMessage = "Unable to connect to Prepar3D:\n" + ex.Message;
            return false;
        }
    }
    else
    {
        errorMessage = "Already connected to SimConnect.";
        return false;
    }
}
}

private IntPtr HandleSimConnectEvents(IntPtr hWnd, int message, IntPtr wParam, IntPtr lParam, ref bool isHandled)
{
    isHandled = false;

    switch (message)
    {
        case WM_USER_SIMCONNECT:
            {
                if (SimConnect != null)
                {
                    try
                    {
                        this.SimConnect.ReceiveMessage();
                    }
                    catch { RecoverFromError(); }

                    isHandled = true;
                }
            }
            break;
        default:
            break;
    }
    return IntPtr.Zero;
}

public void Disconnect()
{
    if (SimConnect != null)
    {
        StopMonitoring();

        // Dispose serves the same purpose as SimConnect_Close()
        this.SimConnect.Dispose();
    }
}

```

```

        this.SimConnect = null;
        OnPropertyChanged("IsConnected");

        SimConnectStatus = "Disconnected from Prepar3D";
        OnPropertyChanged("SimConnectStatus");

        // Clear objects
        Stations.Clear();
        Title = DEFAULT_TITLE;
        NumberOfStationsProcessed = DEFAULT_NUMBER_OF_STATIONS_PROCESSED;
        NumberOfStations = DEFAULT_NUMBER_OF_STATIONS;
        OnPropertyChanged("Title");
        OnPropertyChanged("NumberOfStations");
    }
}

#endregion

public void StartMonitoring()
{
    try
    {
        this.SimConnect.SetSystemEventState(EventIdentifier.Every6Hz, SIMCONNECT_STATE.ON);
    }
    catch { }
}

public void StopMonitoring()
{
    try
    {
        this.SimConnect.SetSystemEventState(EventIdentifier.Every6Hz, SIMCONNECT_STATE.ON);
    }
    catch { }
}

public void QuerySimObjectGeneralQuery()
{
    /**
     * NOTE:
     * Please refer to the Simulation Variables section in the SDK for specific variable names
     */
    try
    {
        this.SimConnect.AddToDataDefinition(DataIdentifier.ReceiveSimObjectGeneral,
            "Title",                                     // Simulation Variable
            null,                                         // Units - for strings put 'null'
            SIMCONNECT_DATATYPE.STRING256,               // Datatype
            0.0f,                                         // Min
            SimConnect.SIMCONNECT_UNUSED);                // Max

        this.SimConnect.AddToDataDefinition(DataIdentifier.ReceiveSimObjectGeneral,
            "Station Count",                           // Simulation Variable
            "number",                                    // Units - unitless "number". Note: use lowercase.
            SIMCONNECT_DATATYPE.FLOAT64,                 // Datatype
            0.0f,                                         // Min
            SimConnect.SIMCONNECT_UNUSED);                // Max

        // IMPORTANT: register it with the simconnect managed wrapper marshaller
        // if you skip this step, you will only receive a uint in the .dwData field.
        this.SimConnect.RegisterDataDefineStruct<SimObjectGeneral>(DataIdentifier.ReceiveSimObjectGeneral);

        this.SimConnect.RequestDataOnSimObjectType(DataIdentifier.ReceiveSimObjectGeneral,
            DataIdentifier.ReceiveSimObjectGeneral,
            0,
            SIMCONNECT_SIMOBJECT_TYPE.USER);
    }
    catch { RecoverFromError(); }
}

public void ReceiveSimObjectGeneralQuery(SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE data)
{
    // Try to cast
    SimObjectGeneral? simObjectGeneral = data.dwData[0] as SimObjectGeneral?;

    if (simObjectGeneral.HasValue)
    {
        // Doesn't check if stations have CHANGED but are the same amount as before.
        if (HaveStationsChanged(simObjectGeneral.Value.Title, simObjectGeneral.Value.NumberOfStations))
        {
            Stations.Clear();
        }

        // Update SimObject display information
        Title = simObjectGeneral.Value.Title;
        OnPropertyChanged("Title");
        NumberOfStations = (int)simObjectGeneral.Value.NumberOfStations;
        OnPropertyChanged("NumberOfStations");

        // Now query all the indices now that we know how many there are
        QueryStationAtAllIndices();
    }
    else
    {
        // Cast failed
    }
}

private bool HaveStationsChanged(string title, double numberOfStations)
{
    return !Title.Equals(title, StringComparison.InvariantCultureIgnoreCase) || numberOfStations != NumberOfStations;
}

public void QueryStationAtAllIndices()
{
    for (int i = 0; i < NumberOfStations; i++)
    {
        QueryStationAtIndex(i);
    }
}

public void QueryStationAtIndex(int i)
{
    try
    {
        this.SimConnect.ClearDataDefinition(DataIdentifier.ReceiveStationAtIndex + i);

        // Register each station with a unique request ID (1 - 1000) is reserved for stations in this example
        this.SimConnect.RegisterDataDefineStruct<StationAtIndex>(DataIdentifier.ReceiveStationAtIndex + i);

        /**
         * NOTE:
         * Please refer to the Simulation Variables section in the SDK for specific variable names
         */

        // Build request definition of what we want returned
        this.SimConnect.AddToDataDefinition(DataIdentifier.ReceiveStationAtIndex + i,
            "Weapon System Station Object Name:" + i, // Simulation Variable which is an array, index it with <Variable Name>:<index>
            null,                                         // Units - for strings put 'null'
            SIMCONNECT_DATATYPE.STRING256,
            0.0f,                                         // Min
            SimConnect.SIMCONNECT_UNUSED);                // Max

        this.SimConnect.AddToDataDefinition(DataIdentifier.ReceiveStationAtIndex + i,
            "Weapon System Station Selected:" + i,      // Simulation Variable which is an array, index it with <Variable Name>:<index>
            "bool",                                       // Units - for strings put 'null'
            SIMCONNECT_DATATYPE.FLOAT64,                  // Datatype (bools can use FLOAT64's)
            0.0f,                                         // Min
            SimConnect.SIMCONNECT_UNUSED);                // Max

        // Request data
        this.SimConnect.RequestDataOnSimObjectType(DataIdentifier.ReceiveStationAtIndex + i, DataIdentifier.ReceiveStationAtIndex + i, 0, SIMCONNECT_SIMOBJECT_TYPE.USER);
    }
    catch { RecoverFromError(); }
}

/// <summary>
/// Method called every time a specific station data query is returned
/// </summary>
/// <param name="data">Data associated with the query </param>
public void ReceiveStationAtIndexQuery(SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE data)
{
}

```

```

        // Try to cast
        StationAtIndex? stationAtIndex = data.dwData[0] as StationAtIndex?;

        if (stationAtIndex.HasValue)
        {
            // TODO: Do I need or not
            // Unregister request ID so it can be used again
            //this.SimConnect.ClearDataDefinition((DataIdentifier)data.dwRequestID);

            Station station = new Station(this);

            // Convert request ID to station ID
            int stationIndex = (int)(data.dwRequestID - (uint)DataIdentifier.RecieveStationAtIndex);

            // Fill data class
            station.Name = stationAtIndex.Value.Title;
            station.Index = stationIndex;
            station.isSelected = stationAtIndex.Value isSelected == 1 ? true : false;
            station.SelectionChanged += new StationSendDataHandler(StationSelectionChanged);

            // Does this station exist?
            int cachedIndex = IndexOfStation(station);

            // Doesn't exist, add it
            if (cachedIndex == -1)
            {
                Stations.Add(station);
            }
            // Does exist. Update it.
            else
            {
                // Is it still being processed?
                if (Stations[cachedIndex].BeingProcessed)
                {
                    // If the expected value matches the returned value (System correctly updated the selected station state)
                    // then we're done processing that station
                    if (Stations[cachedIndex].ExpectedValueForisSelected == station.isSelected)
                    {
                        Stations[cachedIndex].BeingProcessed = false;
                    }
                    // Some extra logic in case SimConnect acts up and doesn't update properly.
                    else
                    {
                        if (Stations[cachedIndex].CurrentProcessRequests < Station.MAX_TIMES_TIL_RESEND)
                        {
                            Stations[cachedIndex].CurrentProcessRequests++;
                        }
                        else // Somehow the system STILL didn't get the request. Resend.
                        {
                            Stations[cachedIndex].CurrentProcessRequests = 0;
                            SetStation(new StationEventArgs(cachedIndex, Stations[cachedIndex].ExpectedValueForisSelected));
                        }
                    }
                }
                // We are free to overwrite the previous station
                else
                {
                    Stations[cachedIndex] = station;
                    OnPropertyChanged("Stations");
                }
            }
            NumberOfStationsProcessed++;
        }

        if (NumberOfStationsProcessed == NumberOfStations)
        {
            // Reset amount of stations processed
            NumberOfStationsProcessed = 0;

            FinishStationQuery();
        }
    }
}

public int IndexOfStation(Station point)
{
    for (int i = 0; i < Stations.Count; i++)
    {
        if (Stations[i].Index == point.Index)
            return i;
    }

    return -1;
}

private void StationSelectionChanged(StationEventArgs stationArgs)
{
    SetStation(stationArgs);
}

public void FinishStationQuery()
{
}

public void SetStation(StationEventArgs station)
{
    try
    {
        this.SimConnect.ClearDataDefinition(DataIdentifier.SendStationAtIndex + station.Index);

        // Build data structure to send to P3D
        this.SimConnect.AddToDataDefinition(DataIdentifier.SendStationAtIndex + station.Index,
            "Weapon System Station Selected:" + station.Index, // Simulation Variable which is an array, index it with <Variable Name>:<index>
            "bool", //bool, // Units - "bool". Note: use lowercase.
            SIMCONNECT_DATATYPE.INT32,
            0.0f,
            SimConnect.SIMCONNECT_UNUSED);

        this.SimConnect.SetDataOnSimObject(DataIdentifier.SendStationAtIndex + station.Index,
            (uint)SIMCONNECT_SIMOBJECT_TYPE.USER,
            SIMCONNECT_DATA_SET_FLAG.DEFAULT,
            station.isSelected ? 1 : 0); // Cast bool to int when working with type "BOOL"
    }
    catch { RecoverFromError(); }
}
}

private void SimConnect_OnRecvSimobjectDataBytype(SimConnect sender, SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE data)
{
    // Must be general SimObject information
    if (data.dwRequestID == (uint)DataIdentifier.RecieveSimObjectGeneral)
    {
        RecieveSimObjectGeneralQuery(data);
    }

    // Must be station data at a specific index
    if (data.dwRequestID >= (uint)DataIdentifier.RecieveStationAtIndex &&
        data.dwRequestID <= (uint)DataIdentifier.RecieveStationEnd)
    {
        RecieveStationAtIndexQuery(data);
    }
}

void SimConnect_OnRecvEvent(SimConnect sender, SIMCONNECT_RECV_EVENT data)
{
    switch (data.uEventID)
    {
        case (uint)EventIdentifier.Every6Hz:
        {
            // Check SimObject properties either to:
            // a) Recieve information for the first time
            // b) Check if station / SimObject information has changed
            QuerySimObjectGeneralQuery();

            break;
        }
    }
}

private void SimConnect_OnRecvException(SimConnect sender, SIMCONNECT_RECV_EXCEPTION data)
{
    //SimConnectStatus = "Exception received: " + data.dwException;
    //OnPropertyChanged("SimConnectStatus");
}

private void SimConnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
{
    this.SimConnect.SetSystemEventState(EventIdentifier.Every6Hz, SIMCONNECT_STATE.OFF);
}

```

```

        SimConnectStatus = "Prepar3D has exited";
        OnPropertyChanged("SimConnectStatus");
    }

    private void SimConnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
    {
        SimConnectStatus = "Connected to Prepar3D";
        OnPropertyChanged("SimConnectStatus");
    }

    private void RecoverFromError()
    {
        string errorMessage;
        Disconnect();

        bool wasSuccess = Connect(out errorMessage);

        // Start monitoring the user's SimObject. This will continuously monitor information
        // about the user's Stations attached to their SimObject.
        if (wasSuccess)
        {
            StartMonitoring();
        }
    }

    #region INotifyPropertyChanged : Allows properties that are bound to tell their controls that they're bound to that they've been updated
    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged(string propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
    #endregion
}

```

```

Station.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Managed_Weapon_Station_Selection
{
    public delegate void StationSendDataHandler(StationEventArgs stationArgs);

    /// <summary>
    /// Create's a data class for returned station information. This will make it possible
    /// to bind the data to a ItemsControl and also to manipulate and compare the data
    /// </summary>
    public class Station
    {
        public const int MAX_TIMES_TIL_RESEND = 10;

        public Station(Managed_Weapon_Station_Selection.DataModel model)
        {
        }

        /// <summary>
        /// The user should have the ability to change whether the station is selected. By changing this
        /// option though, we must set the appropriate SimVar in game which can be done in the callback to
        /// this event.
        /// </summary>
        public event StationSendDataHandler SelectionChanged;

        public string Name { get; set; }
        public int Index { get; set; }

        public bool BeingProcessed { get; set; }
        public int CurrentProcessRequests { get; set; }
        public bool ExpectedValueForIsSelected { get; set; }

        private bool isSelected;
        public bool IsSelected
        {
            get { return isSelected; }
            set
            {
                // If value changed then store the change and
                // send out the change to anybody listening
                if (isSelected != value)
                {
                    isSelected = value;
                    if (Selectionchanged != null)
                    {
                        BeingProcessed = true;
                        CurrentProcessRequests = 0;
                        ExpectedValueForIsSelected = isSelected;
                        SelectionChanged(new StationEventArgs(Index, isSelected));
                    }
                }
            }
        }
    }

    /// <summary>
    /// EventArgs that will be sent along with a Station selection change.
    /// </summary>
    public class StationEventArgs : EventArgs
    {
        public StationEventArgs(int index, bool isSelected)
        {
            Index = index;
            IsSelected = isSelected;
        }

        public int Index { get; private set; }
        public bool IsSelected { get; private set; }
    }
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

ManagedDataRequest.vb

```
'Copyright (c) Lockheed Martin Corporation. All rights reserved.  
'  
' VB.NET Managed Data Request sample  
'  
' Click on Connect to try and connect to a running version of Prepar3D  
' Click on Request Data any number of times  
' Click on Disconnect to close the connection, and then you should  
' be able to click on Connect and restart the process  
'  
  
Imports System  
Imports System.Collections.Generic  
Imports System.ComponentModel  
Imports System.Data  
Imports System.Drawing  
Imports System.Text  
Imports System.Windows.Forms  
  
' Add these two statements to all SimConnect clients  
Imports LockheedMartin.Prepar3D.SimConnect  
Imports System.Runtime.InteropServices  
  
Public Class ManagedDataRequest  
  
    Dim p3d_simconnect As SimConnect  
  
    ' User-defined win32 event  
    Private Const WM_USER_SIMCONNECT As Integer = &H402  
  
    Enum StructDefinitions  
        Struct1 = 1  
    End Enum  
  
    Enum DataRequests  
        Request_1 = 1  
    End Enum  
  
    ' This is how you declare a data structure so that simconnect knows how to fill it/read it.  
  
    <StructureLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Pack:=1)> _  
    Structure Struct1  
        ' This is how you declare a fixed size string  
        <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=256)>_  
        Public title As String  
        Public latitude As Double  
        Public longitude As Double  
        Public altitude As Double  
    End Structure  
    ' Output text line number  
  
    Dim line  
  
    Public Sub New()  
        ' This call is required by the Windows Form Designer.  
        InitializeComponent()  
  
        ' Add any initialization after the InitializeComponent() call.  
        p3d_simconnect = Nothing  
        line = 0  
        SetButtons(True, False, False)  
    End Sub  
  
    Private Sub SetButtons(ByVal bConnect As Boolean, ByVal bRequest As Boolean, ByVal bDisconnect As Boolean)  
        ButtonConnect.Enabled = bConnect  
        ButtonRequest.Enabled = bRequest  
        ButtonDisconnect.Enabled = bDisconnect  
    End Sub  
  
    Private Sub ButtonConnect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles ButtonConnect.Click  
        If p3d_simconnect Is Nothing Then  
            Try  
                p3d_simconnect = New SimConnect("VB Managed Data Request", Me.Handle, WM_USER_SIMCONNECT, Nothing, 0)  
                initDataRequest()  
                DisplayText("Connection request sent ...")  
                SetButtons(False, True, True)  
            Catch ex As Exception  
                DisplayText("Failed to connect")  
            End Try  
        Else  
            DisplayText("Error - try again")  
            closeConnection()  
        End If  
    End Sub  
    Private Sub initDataRequest()  
        ' Set up all the SimConnect related data definitions and event handlers  
        ' listen to connect and quit msgs  
        AddHandler p3d_simconnect.OnRecvOpen, New SimConnect.RecvOpenEventHandler(AddressOf p3d_simconnect_OnRecvOpen)  
        AddHandler p3d_simconnect.OnRecvQuit, New SimConnect.RecvQuitEventHandler(AddressOf p3d_simconnect_OnRecvQuit)  
  
        ' listen to exceptions  
        AddHandler p3d_simconnect.OnRecvException, New SimConnect.RecvExceptionEventHandler(AddressOf p3d_simconnect_OnRecvException)  
  
        ' define a data structure, note the last parameter, datumID must be different for each item  
        p3d_simconnect.AddToDataDefinition(StructDefinitions.Struct1, "title", "", LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.STRING256, 0, 0)  
        p3d_simconnect.AddToDataDefinition(StructDefinitions.Struct1, "Plane Latitude", "degrees", LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.FLOAT64, 0, 1)  
        p3d_simconnect.AddToDataDefinition(StructDefinitions.Struct1, "Plane Longitude", "degrees", LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.FLOAT64, 0, 2)  
        p3d_simconnect.AddToDataDefinition(StructDefinitions.Struct1, "Plane Altitude", "feet", LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_DATATYPE.FLOAT64, 0, 3)  
  
        ' IMPORTANT: register it with the simconnect managed wrapper marshaller  
        ' if you skip this step, you will only receive an int in the .dwData field.  
        p3d_simconnect.RegisterDataDefineStruct(Of Struct1)(StructDefinitions.Struct1)  
  
        ' catch a simobject data request  
        AddHandler p3d_simconnect.OnRecvSimobjectDataBytype, New SimConnect.RecvSimobjectDataBytypeEventHandler(AddressOf p3d_simconnect_OnRecvSimobjectDataBytype)  
    End Sub  
    Private Sub p3d_simconnect_OnRecvOpen(ByVal sender As SimConnect, ByVal data As SIMCONNECT_RECV_OPEN)  
        DisplayText("Connected to Prepar3D")  
    End Sub
```

```

' The case where the user closes Prepar3D
Private Sub p3d_simconnect_OnRecvQuit(ByVal sender As SimConnect, ByVal data As LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_RECV)
    DisplayText("Prepar3D has exited")
    closeConnection()
    SetButtons(True, False, False)

End Sub
Private Sub p3d_simconnect_OnRecvException(ByVal sender As SimConnect, ByVal data As LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_RECV_EXCEPTION)
    DisplayText("Exception received: " + data.dwException.ToString())
End Sub

' The case where the user closes the client
Private Sub Form1_FormClosed(ByVal sender As Object, ByVal e As FormClosedEventArgs) Handles MyBase.FormClosed
    closeConnection()
End Sub
Private Sub p3d_simconnect_OnRecvSimobjectDataBytype(ByVal sender As SimConnect, ByVal data As SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE)
    Select Case data.dwRequestID
        Case DataRequests.Request_1
            ' Cast the data input to the correct structure type
            Dim s1 As Struct1 = CType(data.dwData(0), Struct1)
            DisplayText("title: " + s1.title)
            DisplayText("LIA: " + s1.latitude.ToString("##0.00") + " " + s1.longitude.ToString("##0.00") + " " + s1.altitude.ToString("####0.0"))
        Case Else
            DisplayText("Unknown request ID: " + data.dwRequestID.ToString())
    End Select
End Sub

Private Sub closeConnection()
    If p3d_simconnect IsNot Nothing Then
        p3d_simconnect.Dispose()
        p3d_simconnect = Nothing
        DisplayText("Connection closed")
    End If
End Sub
Private Sub ButtonDisconnect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles ButtonDisconnect.Click
    closeConnection()
    SetButtons(True, False, False)
End Sub
Private Sub ButtonRequest_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles ButtonRequest.Click
    p3d_simconnect.RequestDataOnSimObjectType(DataRequests.Request_1, StructDefinitions.Struct1, 0, LockheedMartin.Prepar3D.SimConnect.SIMCONNECT_SIMOBJECT_TYPE.USER)
    DisplayText("Data request sent ...")
End Sub

Private Sub DisplayText(ByVal text As String)
    Dim output As String
    output = line.ToString() + ": " + text + Environment.NewLine + RichResponses.Text
    RichResponses.Text = output
    line += 1
End Sub
' Simconnect client will send a win32 message when there is
' a packet to process. ReceiveMessage must be called to
' trigger the events. This model keeps simconnect processing on the main thread.
Protected Overrides Sub DefWndProc(ByRef m As Message)
    If m.Msg = WM_USER_SIMCONNECT Then
        If p3d_simconnect IsNot Nothing Then
            p3d_simconnect.ReceiveMessage()
        End If
    Else
        MyBase.DefWndProc(m)
    End If
End Sub
End Class

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
 The names of the actual companies and products mentioned
 herein may be trademarks of their respective owners.

PREPAR3D

DataHarvester.cpp

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.  
//-----  
// SimConnect Data Harvester Sample  
//  
// Description:  
// Requests data on the user object and outputs it to a CSV file.  
// If the sample application is ran from within the SimConnect  
// Samples solution, the CSV file will be created relative to the  
// DataHarvester project file. If the sample application is executed  
// directly, the CSV file will be created relative to the application.  
// Start and stop data harvesting from the Prepar3D Add-ons menu item.  
//-----  
  
#include <Windows.h>  
#include <uchar.h>  
#include <strsafe.h>  
  
#include "SimConnect.h"  
  
int g_bQuit = 0;  
HANDLE g_hSimConnect = NULL;  
FILE* g_pFile = NULL;  
bool g_bIsHarvesting = false;  
  
static const char* TITLE_STRING = "Data Harvester";  
static const char* START_STRING = "Start Harvest (SimConnect)";  
static const char* STOP_STRING = "Stop Harvest (SimConnect)";  
  
static const char* OUTPUT_FILE_NAME = "DataHarvester.csv";  
  
enum GROUP_ID  
{  
    GROUP_ID_MENU,  
};  
  
enum EVENT_ID  
{  
    EVENT_ID_START_HARVEST,  
    EVENT_ID_STOP_HARVEST,  
};  
  
enum DEFINITION_ID  
{  
    DEFINITION_ID_USER_OBJECT,  
};  
  
enum REQUEST_ID  
{  
    REQUEST_ID_USER_OBJECT_DATA,  
};  
  
// This struct must align with the sizes and types defined in the table below.  
struct ObjectData  
{  
    char szTitle[256];  
    double dAbsoluteTime;  
    double dTime;  
    int uDayOfTheYear;  
    int uYear;  
    int uDayOfTheMonth;  
    int uDayOfTheWeek;  
    int uTimezoneOffset;  
    double dsimTime;  
    double dLatitude;  
    double dLongitude;  
    double dAltitude;  
    double dPitch;  
    double dBank;  
    double dHeading;  
    double dVelocityX;  
    double dVelocityY;  
    double dVelocityZ;  
    double dTemperature;  
    double dAlphaBeta;  
    double dAirDensity;  
    double dWindVelocity;  
    double dWindDirection;  
    double dWindX;  
    double dWindY;  
    double dWindZ;  
};  
  
struct PropertyDefinition  
{  
    const char* pszName;  
    const char* pszUnits;  
    SIMCONNECT_DATATYPE eDataType;  
};  
  
// This table must align with the sizes and types defined in the struct above.  
const PropertyDefinition g_aVariables[] =  
{  
    {"TITLE", "", NULL, SIMCONNECT_DATATYPE_STRING256},  
    {"ABSOLUTE TIME", "Seconds", SIMCONNECT_DATATYPE_FLOAT64},  
    {"ZULU TIME", "Seconds", SIMCONNECT_DATATYPE_FLOAT64},  
    {"ZULU DAY OF YEAR", "Number", SIMCONNECT_DATATYPE_INT32},  
    {"ZULU YEAR", "Number", SIMCONNECT_DATATYPE_INT32},  
    {"ZULU MONTH OF YEAR", "Number", SIMCONNECT_DATATYPE_INT32},  
    {"ZULU DAY OF MONTH", "Number", SIMCONNECT_DATATYPE_INT32},  
    {"ZULU DAY OF WEEK", "Number", SIMCONNECT_DATATYPE_INT32},  
    {"TIME ZONE OFFSET", "Hours", SIMCONNECT_DATATYPE_INT32},  
    {"WORLD TIME", "Seconds", SIMCONNECT_DATATYPE_FLOAT64},  
    {"PLANE LATITUDE", "Degrees", SIMCONNECT_DATATYPE_FLOAT64},  
    {"PLANE LONGITUDE", "Degrees", SIMCONNECT_DATATYPE_FLOAT64},  
    {"PLANE ALTITUDE", "Feet", SIMCONNECT_DATATYPE_FLOAT64},  
    {"PLANE PITCH DEGREES", "Degrees", SIMCONNECT_DATATYPE_FLOAT64},  
    {"PLANE BANK DEGREES", "Degrees", SIMCONNECT_DATATYPE_FLOAT64},  
    {"PLANE HEADING DEGREES TRUE", "Degrees", SIMCONNECT_DATATYPE_FLOAT64},  
    {"VELOCITY X", "Meters per second", SIMCONNECT_DATATYPE_FLOAT64},  
    {"VELOCITY WORLD Y", "Feet per second", SIMCONNECT_DATATYPE_FLOAT64},  
    {"VELOCITY WORLD Z", "Feet per second", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT TEMPERATURE", "Celsius", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT PRESSURE", "Millibars", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT DENSITY", "Slugs per cubic feet", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT WIND VELOCITY", "Knots", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT WIND DIRECTION", "Degrees", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT WIND X", "Meters per second", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT WIND Y", "Meters per second", SIMCONNECT_DATATYPE_FLOAT64},  
    {"AMBIENT WIND Z", "Meters per second", SIMCONNECT_DATATYPE_FLOAT64},  
};  
  
void PrintHeader()  
{  
    char szBuffer[2048] = { 0 };  
  
    for (unsigned int i = 0; i < ARRAYSIZE(g_aVariables); ++i)  
    {  
        const PropertyDefinition* prop = g_aVariables[i];  
  
        strcat_s(szBuffer, sizeof(szBuffer), prop.pszName);  
  
        if (prop.pszUnits)  
        {  
            char szUnits[64] = { 0 };  
            StringCchPrintfA(szUnits, sizeof(szUnits), " (%s)", prop.pszUnits);  
            strcat_s(szBuffer, sizeof(szBuffer), szUnits);  
        }  
  
        if (i != ARRAYSIZE(g_aVariables) - 1)  
        {  
            strcat_s(szBuffer, sizeof(szBuffer), ",");  
        }  
    }  
}
```

```

        strcat_s(szBuffer, sizeof(szBuffer), "\n");
        fprintf_s(g_pFile, szBuffer);
    }

void PrintString(char* pszBuffer, unsigned int cbBuffer, const char* pszValue, bool bIsLast = false)
{
    if (pszBuffer != NULL && cbBuffer > 0)
    {
        char szTemp[256] = { 0 };
        if (!bIsLast)
        {
            if (pszValue)
            {
                StringCchPrintfA(szTemp, sizeof(szTemp), "%s", pszValue);
                strcat_s(pszBuffer, cbBuffer, szTemp);
            }
            else
            {
                StringCchPrintfA(szTemp, sizeof(szTemp), "", pszValue);
                strcat_s(pszBuffer, cbBuffer, szTemp);
            }
        }
        else
        {
            if (pszValue)
            {
                StringCchPrintfA(szTemp, sizeof(szTemp), "%s\n", pszValue);
                strcat_s(pszBuffer, cbBuffer, szTemp);
            }
            else
            {
                StringCchPrintfA(szTemp, sizeof(szTemp), "\n", pszValue);
                strcat_s(pszBuffer, cbBuffer, szTemp);
            }
        }
    }
}

void PrintInt(char* pszBuffer, unsigned int cbBuffer, int iValue, bool bIsLast = false)
{
    if (pszBuffer != NULL && cbBuffer > 0)
    {
        char szTemp[128] = { 0 };
        if (!bIsLast)
        {
            StringCchPrintfA(szTemp, sizeof(szTemp), "%d", iValue);
            strcat_s(pszBuffer, cbBuffer, szTemp);
        }
        else
        {
            StringCchPrintfA(szTemp, sizeof(szTemp), "%d\n", iValue);
            strcat_s(pszBuffer, cbBuffer, szTemp);
        }
    }
}

void PrintDouble(char* pszBuffer, unsigned int cbBuffer, double dValue, bool bIsLast = false)
{
    if (pszBuffer != NULL && cbBuffer > 0)
    {
        char szTemp[128] = { 0 };
        if (!bIsLast)
        {
            StringCchPrintfA(szTemp, sizeof(szTemp), "%.20f", dValue);
            strcat_s(pszBuffer, cbBuffer, szTemp);
        }
        else
        {
            StringCchPrintfA(szTemp, sizeof(szTemp), "%.20f\n", dValue);
            strcat_s(pszBuffer, cbBuffer, szTemp);
        }
    }
}

void CALLBACK MyDispatchProcRD(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    HRESULT hr;
    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;
            switch (evt->uEventID)
            {
                case EVENT_ID_START_HARVEST:
                {
                    // Verify the file has been opened. Notify the user otherwise.
                    printf("\nAttempting to open CSV file...");
                    if (g_pfile)
                    {
                        fclose(g_pfile);
                        g_pfile = NULL;
                    }
                    fopen_s(&g_pFile, OUTPUT_FILE_NAME, "w");
                    if (!g_pFile)
                    {
                        printf("Failed to open \"%s\". Please ensure the file has the correct permissions and is not being used by another application.", OUTPUT_FILE_NAME);
                        break;
                    }
                    printf("\nFile successfully opened!");

                    g_bIsHarvesting = true;

                    printf("\nStarting data harvest...");

                    // Print variable titles into the CSV.
                    PrintHeader();

                    // Perform data request on the user object.
                    hr = SimConnect_RequestDataOnSimObject(g_hSimConnect, REQUEST_ID_USER_OBJECT_DATA, DEFINITION_ID_USER_OBJECT, SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD_VISUAL_FRAME);

                    // Update menu items.
                    hr = SimConnect_MenuDeleteItem(g_hSimConnect, EVENT_ID_START_HARVEST);
                    hr = SimConnect_MenuAddItem(g_hSimConnect, STOP_STRING, EVENT_ID_STOP_HARVEST, 0);

                    break;
                }
                case EVENT_ID_STOP_HARVEST:
                {
                    // Cancel the data request on the user object.
                    hr = SimConnect_RequestDataOnSimObject(g_hSimConnect, REQUEST_ID_USER_OBJECT_DATA, DEFINITION_ID_USER_OBJECT, SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD_NEVER);

                    // Update menu items.
                    hr = SimConnect_MenuDeleteItem(g_hSimConnect, EVENT_ID_STOP_HARVEST);
                    hr = SimConnect_MenuAddItem(g_hSimConnect, START_STRING, EVENT_ID_START_HARVEST, 0);

                    // Close the file.
                    if (g_pfile)
                    {
                        fclose(g_pFile);
                        g_pfile = NULL;
                    }

                    printf("\nStopping data harvest.");

                    g_bIsHarvesting = false;

                    break;
                }
                default:
                {
                    break;
                }
            }
        }
        break; // SIMCONNECT_RECV_ID_EVENT
    }

    case SIMCONNECT_RECV_ID_SIMOBJECT_DATA:
    {
        SIMCONNECT_RECV_SIMOBJECT_DATA* pObjData = (SIMCONNECT_RECV_SIMOBJECT_DATA*)pData;
        switch(pObjData->dwRequestId)
        {
            case REQUEST_ID_USER_OBJECT_DATA:
            {
                DWORD dwObjectID = pObjData->dwObjectId;
            }
        }
    }
}

```

```

ObjectData* pUserData = (ObjectData*)&pObjData->dwData;
    // Console print a portion of the data.
    if (SUCCEEDED(StringCchLengthA(&pUserData->szTitle[0], sizeof(pUserData->szTitle), NULL))) // security check
    {
        printf("\nData: Time=%f Lat=%f Lon=%f Alt=%f",
            pUserData->dTime,
            pUserData->dLatitude,
            pUserData->dLongitude,
            pUserData->dAltitude);
    }

    if (g_pFile != NULL)
    {
        char szBuffer[4096] = { 0 };

        // Stream the user object's data into the CSV file.
        PrintString(szBuffer, sizeof(szBuffer), pUserData->szTitle);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dAbsoluteTime);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dTime);
        PrintInt(szBuffer, sizeof(szBuffer), pUserData->dDayOfTheYear);
        PrintInt(szBuffer, sizeof(szBuffer), pUserData->dYear);
        PrintInt(szBuffer, sizeof(szBuffer), pUserData->dMonthOfTheYear);
        PrintInt(szBuffer, sizeof(szBuffer), pUserData->dDayOfTheMonth);
        PrintInt(szBuffer, sizeof(szBuffer), pUserData->dDayOfTheWeek);
        PrintInt(szBuffer, sizeof(szBuffer), pUserData->dTimeZoneOffset);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dSineTransform);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dLatitude);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dLongitude);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dAltitude);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dPitch);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dBank);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dHeading);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dVelocityX);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dVelocityY);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dVelocityZ);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dTemperture);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dAirPressure);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dAirDensity);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dWindVelocity);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dWindDirection);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dWindX);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dWindY);
        PrintDouble(szBuffer, sizeof(szBuffer), pUserData->dWindZ, true);
        fprintf_s(g_pFile, szBuffer);
    }
    break;
}
default:
{
    break;
}
}

break; // SIMCONNECT_RECV_ID_SIMOBJECT_DATA
}

case SIMCONNECT_RECV_ID_QUIT:
{
    g_bQuit = 1;
    break;
}

default:
{
    printf("\nReceived: %d", pData->dwID);
    break;
}
}

void RunDataHarvester()
{
    HRESULT hr;

    bool bConnected = false;

    // Attempt to connect to Prepar3D.
    printf("\nAttempting to connect to Prepar3D...");
    for (unsigned int i = 0; i < 60; ++i)
    {
        if (SUCCEEDED(SimConnect_Open(&g_hSimConnect, TITLE_STRING, NULL, 0, 0)))
        {
            bConnected = true;
            break;
        }
        printf("\nAttempt %d", i + 1);
        Sleep(1000);
    }

    if (bConnected)
    {
        printf("\nConnected to Prepar3D!");

        // Set up the data definition.
        for (unsigned int i = 0; i < ARRAYSIZE(g_aVariables); ++i)
        {
            const PropertyDefinition& prop = g_aVariables[i];
            hr = SimConnect_AddObjectDefinition(g_hSimConnect, DEFINITION_ID_USER_OBJECT, prop.pszName, prop.pszUnits, prop.eDataType);
        }

        // Add the menu item that will start the harvest.
        hr = SimConnect_MenuAddItem(g_hSimConnect, START_STRING, EVENT_ID_START_HARVEST, 0);

        // Application main loop.
        while (0 == g_bQuit)
        {
            SimConnect_CallDispatch(g_hSimConnect, MyDispatchProcRD, NULL);
            Sleep(1);
        }

        // Clean up.
        if (g_bIsHarvesting)
        {
            hr = SimConnect_RequestDataOnSimObject(g_hSimConnect, REQUEST_ID_USER_OBJECT_DATA, DEFINITION_ID_USER_OBJECT, SIMCONNECT_OBJECT_ID_USER, SIMCONNECT_PERIOD_NEVER);
            hr = SimConnect_MenuDeleteItem(g_hSimConnect, EVENT_ID_STOP_HARVEST);

            if (g_pFile)
            {
                fclose(g_pFile);
                g_pFile = NULL;
            }
            else
            {
                hr = SimConnect_MenuDeleteItem(g_hSimConnect, EVENT_ID_START_HARVEST);
            }
        }

        // Close.
        hr = SimConnect_Close(g_hSimConnect);
    }
    else
    {
        printf("\nConnection timeout!");
    }
}

int __cdecl _tmain(int argc, _TCHAR* argv[])
{
    RunDataHarvester();
    return 0;
}

```

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

ScenarioController.cs

```
//Copyright (c) Lockheed Martin Corporation. All rights reserved.
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Managed_Scenario_Controller
{
    public class ScenarioController
    {
        private enum Selection
        {
            Quit = 0,
            LaunchScenario = 1,
            Max,
        };

        private const string SubKeyString = @"Software\Lockheed Martin\Prepar3D v5";
        private const string ExecutableFilename = "Prepar3D.exe";

        private String mSetupPath = ""; // The Prepar3D application directory.
        private String mExecutableFilePath = ""; // The Prepar3D.exe file path.
        private String mScenarioFilename = ""; // The filename of the scenario to load.

        // Boolean to determine if the application should quit.
        private bool mQuit = false;

        // Boolean representing whether or not the Prepar3D process is running.
        private bool mProcRunning = false;

        public void Run()
        {
            System.Console.WriteLine("Prepar3D Scenario Controller Sample\n");

            // Attempt to retrieve the Prepar3D application directory from the registry.
            bool validSetupPath = LookupSetupPath();

            // If we fail to locate it from the registry, prompt the user for the location.
            if (!validSetupPath)
            {
                validSetupPath = PromptForSetupPath();
            }

            // Continue if the provided path is valid.
            if (validSetupPath)
            {
                // Confirm the executable exists.
                bool validExecutablePath = InitializeExecutablePath();
                if (validExecutablePath)
                {
                    // User selection loop until they quit.
                    while (!mQuit)
                    {
                        // Get the selected option from the user.
                        Selection selection = PromptForSelection();
                        switch(selection)
                        {
                            case Selection.LaunchScenario:
                            {
                                // Get the scenario file name from the user.
                                if (PromptForScenario())
                                {
                                    // Launch the Prepar3D application with the selected scenario.
                                    LaunchScenario();

                                    // Confirm the application successfully started.
                                    if (!mProcRunning)
                                    {
                                        // Create and start the SimConnect based scenario data provider.
                                        ScenarioDataProvider DataProvider = new ScenarioDataProvider();
                                        if (DataProvider.Start())
                                        {
                                            // Wait for the Prepar3D process to exit.
                                            while (!mProcRunning)
                                            {
                                                Thread.Sleep(1000);
                                            }
                                        }
                                    }
                                    mProcRunning = false;
                                }
                            }
                            break;
                            case Selection.Quit:
                            {
                                mQuit = true;
                                break;
                            }
                        }
                    }
                }
            }

            // Leave the console open to display errors if the user didn't manually quit.
            if (!mQuit)
            {
                System.Console.WriteLine("Press enter to exit the application...");
                System.Console.ReadLine();
            }
        }

        private void LaunchScenario()
        {
            ProcessStartInfo startInfo = new ProcessStartInfo();
            startInfo.FileName = mExecutableFilePath;
            startInfo.Arguments = "\"" + mScenarioFilename + "\"";
            System.Console.WriteLine(String.Format("Attempting to launch scenario: \"{0}\", {1}", mScenarioFilename));
        }
    }
}
```

```

        try
        {
            Process proc = Process.Start(startInfo);
            proc.EnableRaisingEvents = true;
            proc.Exited += Proc_Exited;
            mProcRunning = true;
            System.Console.WriteLine("Prepar3D process started.");
        }
        catch
        {
            System.Console.WriteLine("ERROR: Failed to start Prepar3D process.");
        }
    }

    private void Proc_Exited(object sender, EventArgs e)
    {
        System.Console.WriteLine("Prepar3D process closed.");
        System.Console.WriteLine("");
        mProcRunning = false;
    }

    private Selection PromptForSelection()
    {
        Selection result = Selection.Quit;

        while (true)
        {
            System.Console.WriteLine("Select an option:");
            System.Console.WriteLine("1. Launch Scenario");
            System.Console.WriteLine("0. Quit");

            uint selection = 0;
            String userInput = System.Console.ReadLine();
            System.Console.WriteLine("");
            if (String.IsNullOrEmpty(userInput) || !UInt32.TryParse(userInput, out selection) || selection >= (uint)Selection.Max)
            {
                System.Console.WriteLine("ERROR: Invalid option.");
            }
            else
            {
                result = (Selection)selection;
                break;
            }
        }

        return result;
    }

    private bool PromptForScenario()
    {
        bool result = false;

        string scenarioFilename = "";

        while (String.IsNullOrEmpty(scenarioFilename))
        {
            System.Console.WriteLine("Enter the filename of the scenario to launch:");
            scenarioFilename = System.Console.ReadLine();
            System.Console.WriteLine("");
            if (String.IsNullOrEmpty(scenarioFilename))
            {
                System.Console.WriteLine("ERROR: Please enter a valid scenario filename.");
            }
        }

        mScenarioFilename = scenarioFilename;
        result = true;

        return result;
    }

    private bool InitializeExecutablePath()
    {
        bool result = false;

        string executableFilePath = mSetupPath + ExecutableFilename;

        if (File.Exists(executableFilePath))
        {
            mExecutableFilePath = executableFilePath;
            result = true;
        }
        else
        {
            System.Console.WriteLine(String.Format("ERROR: Failed to locate \'{0}\'.", executableFilePath));
        }

        return result;
    }

    private bool LookupSetupPath()
    {
        bool result = false;

        // Force the 64-bit version location.
        Microsoft.Win32.RegistryKey BaseKey = Microsoft.Win32.RegistryKey.OpenBaseKey(Microsoft.Win32.RegistryHive.LocalMachine,
                                                                                           Microsoft.Win32.RegistryView.Registry64);

        if (BaseKey != null)
        {
            Microsoft.Win32.RegistryKey SubKey = BaseKey.OpenSubKey(SubKeyString);
            if (SubKey != null)
            {
                string setupPath = SubKey.GetValue("SetupPath") as string;
                if (!String.IsNullOrEmpty(setupPath))
                {
                    if (Directory.Exists(setupPath))
                    {
                        mSetupPath = setupPath;
                        result = true;
                    }
                }
            }
        }

        if (!result)
        {
            System.Console.WriteLine("WARNING: Failed to lookup Prepar3D installation directory from registry.");
        }

        return result;
    }

    private bool PromptForSetupPath()
    {
        bool result = false;

        System.Console.WriteLine("Enter the installation directory of Prepar3D:");
        string setupPath = "";

```

```

        while (!Directory.Exists(setupPath))
        {
            setupPath = System.Console.ReadLine();
            if (!Directory.Exists(setupPath))
            {
                System.Console.WriteLine(String.Format("ERROR: The directory \'{0}\' does not exist. Please enter a valid directory.", setupPath));
            }
        }

        mSetupPath = setupPath;
        result = true;
    }
}

```

ScenarioDataProvider.cs

```

//Copyright (c) Lockheed Martin Corporation. All rights reserved.
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

// Add these two statements to all SimConnect clients
using LockheedMartin.Prepar3D.SimConnect;
using System.Runtime.InteropServices;

namespace Managed_Scenario_Controller
{
    public class ScenarioDataProvider
    {
        /// <summary>
        /// SimConnect Enum for Mission Completion Status types
        /// </summary>
        enum MissionStatus
        {
            MissionStatusFailed,
            MissionStatusCrashed,
            MissionStatusSucceeded,
        };

        public enum EventIdentifier
        {
            MissionCompleted = 0,
        }

        private SimConnect SimConnect = null;
        private Thread SimConnectThread = null;
        private bool mQuit = false;

        public ScenarioDataProvider()
        {
        }

        /// <summary>
        /// Starts the SimConnect based thread.
        /// </summary>
        public bool Start()
        {
            bool result = false;

            try
            {
                SimConnectThread = new Thread(new ThreadStart(StartSimConnectThread));
                SimConnectThread.IsBackground = true;
                SimConnectThread.Start();
                result = true;
            }
            catch
            {
            }

            return result;
        }

        /// <summary>
        /// Attempt to create the SimConnect connection with Prepar3D.
        /// </summary>
        private bool PollForConnection()
        {
            int retryCounter = 1000;

            while (SimConnect == null && retryCounter > 0)
            {
                try
                {
                    SimConnect = new SimConnect("Managed Scenario Controller", IntPtr.Zero, 0, null, 0);
                    if (SimConnect != null)
                    {
                        SimConnect.OnRecvOpen += new LockheedMartin.Prepar3D.SimConnect.SimConnect.RecvOpenEventHandler(SimConnect_OnRecvOpen);
                        SimConnect.OnRecvQuit += new LockheedMartin.Prepar3D.SimConnect.SimConnect.RecvQuitEventHandler(SimConnect_OnRecvQuit);
                        SimConnect.OnRecvEvent += new LockheedMartin.Prepar3D.SimConnect.SimConnect.RecvEventEventHandler(SimConnect_OnRecvEvent);

                        // Subscribe to system events
                        SimConnect.SubscribeToSystemEvent(EventIdentifier.MissionCompleted, "MissionCompleted");

                        // Turn the events on
                        SimConnect.SetSystemEventState(EventIdentifier.MissionCompleted, SIMCONNECT_STATE.ON);
                    }
                }
                catch
                {
                }

                if (SimConnect == null)
                {
                    Thread.Sleep(500);
                    --retryCounter;
                }
            }

            return (retryCounter > 0);
        }

        /// <summary>
        /// Creates the SimConnect connection with Prepar3D and
        /// continuously requests to receive messages.
        /// </summary>
        private void StartSimConnectThread()
        {
            Thread.Sleep(2500);
        }
    }
}

```

```

        if (PollForConnection())
        {
            if (SimConnect != null)
            {
                while (!mQuit)
                {
                    try
                    {
                        SimConnect.ReceiveMessage();
                    }
                    catch
                    {
                    }

                    Thread.Sleep(500);
                }

                if (mQuit)
                {
                    SimConnect.Dispose();
                    SimConnect = null;
                }
            }
        }
        else
        {
            System.Console.WriteLine("ERROR: SimConnect failed to connect (timed out).");
        }
    }

    /// <summary>
    /// Callback method for initial Receive-Open SimConnect operation
    /// </summary>
    /// <param name="sender">SimConnect object initiating the callback</param>
    /// <param name="data">Receive-Open SimConnect data object associated with the event</param>
    private void SimConnect_OnRecvOpen(SimConnect sender, SIMCONNECT_RECV_OPEN data)
    {
        System.Console.WriteLine("SimConnect connected to Prepar3D.");
    }

    /// <summary>
    /// Callback method for initial Receive-Quit SimConnect operation
    /// </summary>
    /// <param name="sender">SimConnect object initiating the callback</param>
    /// <param name="data">Receive-Quit SimConnect data object associated with the event</param>
    private void SimConnect_OnRecvQuit(SimConnect sender, SIMCONNECT_RECV data)
    {
        System.Console.WriteLine("SimConnect disconnected from Prepar3D.");
        mQuit = true;
    }

    /// <summary>
    /// Callback method for Receive-Event SimConnect operation
    /// </summary>
    /// <param name="sender">SimConnect object initiating the callback</param>
    /// <param name="data">Receive-Event SimConnect data object associated with the event</param>
    void SimConnect_OnRecvEvent(SimConnect sender, SIMCONNECT_RECV_EVENT data)
    {
        switch ((EventIdentifier)data.uEventID)
        {
            // Mission completion result.
            // This type of scenario information could potentially be used by an LMS,
            // however is beyond the scope of this sample.
            case EventIdentifier.MissionCompleted:
            {
                switch ((MissionStatus)data.dwData)
                {
                    case MissionStatus.MissionStatusFailed:
                    {
                        System.Console.WriteLine("Mission completion status: FAILED.");
                        break;
                    }
                    case MissionStatus.MissionStatusCrashed:
                    {
                        System.Console.WriteLine("Mission completion status: CRASHED.");
                        break;
                    }
                    case MissionStatus.MissionStatusSucceeded:
                    {
                        System.Console.WriteLine("Mission completion status: SUCCEEDED.");
                        break;
                    }
                }
                break;
            }
        }
    }
}

```

Prepar3D Development (PDK) API

Related Links

- [SDK Overview](#)

Overview

The Prepar3D Development Kit (PDK) API is a service provider for obtaining services to the Prepar3D platform. The PDK functions similarly to SimConnect but is tied into Prepar3D at a lower level, allowing better performance and more direct interaction. Unlike SimConnect, there is not a network interface for PDK plug-ins as they must be developed as in-process DLLs. Data can be injected into and received from the simulation by interfacing with a user-written DLL. Uses for the PDK include creating custom SimObjects, adding custom textures and post-processes, and modifying cameras.

The following list contains service providers and development samples within Prepar3D:

- Service Providers
 - [Configuration Service](#)
 - [Data Load Helper](#)
 - [Engine System](#)
 - [Environment Force Service](#)
 - [Event Service](#)
 - [Global Data Service](#)
 - [Icon Service](#)
 - [Library Object Data](#)
 - [Menu Service](#)
 - [Network Services](#)
 - [Panel System Service](#)
 - [Rendering Services](#)
 - [Reporting Service](#)
 - [Scenario Manager Service](#)
 - [Sim Property Service](#)
 - [SimObject Services](#)
 - [Sound Service](#)
 - [Virtual Reality Service](#)
 - [Visual Effects Service](#)
 - [Voice Control Service](#)
 - [Weather System Service](#)
 - [Window and Camera Services](#)
 - [World Object Service](#)
 - PDK Interface Wrappers and Types
 - [PDK Services](#)
 - [PDK Types](#)
 - Samples
 - [Samples Overview](#)
 - [Camera Sample](#)
 - [Camera Picking Sample](#)
 - [Custom Lights Sample](#)
 - [Custom PDK Objects](#)
 - [Radar Sample](#)
 - [Targeting Pod Sample](#)
 - [Texture and Effect Render Plug-in Gauge Sample](#)
-

[- top -](#)

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

PDK

Overview

Classes

struct **P3DDXYZ**

Simple vector structures for passing between **P3D** host and implementation. [More...](#)

struct **P3DFXYZ**

class **IPdk**

class **IPdkV01**

Current version of the Prepar3d Sdk. [More...](#)

class **PdkPlugin::P3dPluginCallback**

class **PdkPlugin::OneHzCallback**

class **PdkPlugin::FrameCallback**

class **PdkPlugin::CustomRenderCallback**

class **PdkPlugin::MessageCallback**

class **PdkPlugin**

class **PdkServices**

Class Documentation

◆ **P3D::P3DDXYZ**

struct P3D::P3DDXYZ

Simple vector structures for passing between **P3D** host and implementation.

[Class Members](#)

double dX

double	dY	
double	dZ	

◆ P3D::P3DFXYZ

struct P3D::P3DFXYZ

Class Members		
float	fX	
float	fY	
float	fZ	

◆ P3D::IPdk

class P3D::IPdk

IPdk - The primary interface to the Prepar3D SDK (PDK)

- This is a refcounted object which should be released when no longer needed
- The primary function of this is QueryService which will provide the necessary services.

Inherits IServiceProvider.

Inherited by **IPdkV01**.

Private Member Functions

virtual HRESULT **RegisterService** (REFGUID guidService, IUnknown *punkService) **PURE**
Registers an IUnknown service that can be queried by consumers of the
SDK. [More...](#)

virtual HRESULT **UnRegisterService** (REFGUID guidService) **PURE**
UnRegisters an IUnknown service when it is no longer available. e.g.
shutdown. [More...](#)

Member Function Documentation

◆ RegisterService()

```
virtual HRESULT RegisterService ( REFGUID guidService,
                                 IUnknown * punkService
                               )
```

private
virtual

Registers an IUnknown service that can be queried by consumers of the SDK.

◆ UnRegisterService()

```
virtual HRESULT UnRegisterService ( REFGUID guidService )
```

private
virtual

UnRegisters an IUnknown service when it is no longer available. e.g. shutdown.

◆ P3D::IPdkV01

class P3D::IPdkV01

Current version of the Prepar3d Sdk.

Inherits [IPdk](#).

◆ P3D::PdkPlugin::P3dPluginCallback

class P3D::PdkPlugin::P3dPluginCallback

Helper class that defines a callback for a specific event ID

Inherits [ICallbackV400](#).

Inherited by [PdkPlugin::CustomRenderCallback](#), [PdkPlugin::FrameCallback](#), [PdkPlugin::MessageCallback](#), and [PdkPlugin::OneHzCallback](#).

Public Member Functions

```
P3dPluginCallback (PdkPlugin *pPlugin, const GUID &eventID)
void Register (IEventServiceV400 *pEventService)
void Unregister (IEventServiceV400 *pEventService)
virtual void Invoke (IParameterListV400 *pParams) override
    DEFAULT_REFCOUNT_INLINE_IMPL ()
STDMETHODIMP QueryInterface (REFIID riid, PVOID *ppv)
```

Public Attributes

```
const GUID & m_EventID
PdkPlugin * m_pPlugin
```

Constructor & Destructor Documentation

◆ P3dPluginCallback()

```
P3dPluginCallback ( PdkPlugin * pPlugin,
                    const GUID & eventID
                )
```

inline

Member Function Documentation

◆ DEFAULT_REFCOUNT_INLINE_IMPL()

```
DEFAULT_REFCOUNT_INLINE_IMPL( )
```

◆ Invoke()

```
virtual void Invoke ( IParameterListV400 * pParams ) [pure virtual]
```

Called when the registered event is fired.

Parameters

pParams Callback parameters interface pointer.

Implements **ICallbackV400**.

Implemented in **PdkPlugin::MessageCallback**, **PdkPlugin::CustomRenderCallback**, **PdkPlugin::FrameCallback**, and **PdkPlugin::OneHzCallback**.

◆ QueryInterface()

```
STDMETHODIMP QueryInterface ( REFIID riid,  
                           PVOID * ppv  
                         ) [inline]
```

◆ Register()

```
void Register ( IEventServiceV400 * pEventService ) [inline]
```

Register this callback

◆ Unregister()

```
void Unregister ( IEventServiceV400 * pEventService ) [inline]
```

Unregister this callback

Member Data Documentation

◆ m_EventID

```
const GUID& m_EventID
```

◆ m_pPlugin

PdkPlugin* m_pPlugin

◆ P3D::PdkPlugin::OneHzCallback

class P3D::PdkPlugin::OneHzCallback

Callback class that calls OnOneHz

Inherits **PdkPlugin::P3dPluginCallback**.

Public Member Functions

OneHzCallback (PdkPlugin *pPlugin)

virtual void **Invoke (IParame**terListV400 *pParams) override

▶ Public Member Functions inherited from **PdkPlugin::P3dPluginCallback**

Additional Inherited Members

▶ Public Attributes inherited from **PdkPlugin::P3dPluginCallback**

Constructor & Destructor Documentation

◆ OneHzCallback()

OneHzCallback (PdkPlugin * pPlugin) **inline**

Member Function Documentation

◆ Invoke()

virtual void **Invoke (IParameterListV400 * pParams)** **inline** **override** **virtual**

Called when the registered event is fired.

Parameters

pParams Callback parameters interface pointer.

Implements **PdkPlugin::P3dPluginCallback**.

◆ P3D::PdkPlugin::FrameCallback

class P3D::PdkPlugin::FrameCallback

Callback class that calls OnFrame

Inherits **PdkPlugin::P3dPluginCallback**.

Public Member Functions

FrameCallback (`PdkPlugin *pPlugin`)
virtual void `Invoke (IParameterListV400 *pParams)` override

▶ Public Member Functions inherited from `PdkPlugin::P3dPluginCallback`

Additional Inherited Members

▶ Public Attributes inherited from `PdkPlugin::P3dPluginCallback`

Constructor & Destructor Documentation

◆ `FrameCallback()`

`FrameCallback (PdkPlugin * pPlugin)` [inline](#)

Member Function Documentation

◆ `Invoke()`

virtual void `Invoke (IParameterListV400 * pParams)` [inline](#) [override](#) [virtual](#)

Called when the registered event is fired.

Parameters

`pParams` Callback parameters interface pointer.

Implements `PdkPlugin::P3dPluginCallback`.

◆ `P3D::PdkPlugin::CustomRenderCallback`

class `P3D::PdkPlugin::CustomRenderCallback`

Callback class that calls `OnCustomRender`

Inherits `PdkPlugin::P3dPluginCallback`.

Public Member Functions

CustomRenderCallback (`PdkPlugin *pPlugin`)
virtual void `Invoke (IParameterListV400 *pParams)` override

▶ Public Member Functions inherited from `PdkPlugin::P3dPluginCallback`

Additional Inherited Members

▶ Public Attributes inherited from `PdkPlugin::P3dPluginCallback`

Constructor & Destructor Documentation

◆ CustomRenderCallback()

CustomRenderCallback (**PdkPlugin** * **pPlugin**) [inline](#)

Member Function Documentation

◆ Invoke()

virtual void Invoke (**IParameterListV400** * **pParams**) [inline](#) [override](#) [virtual](#)

Called when the registered event is fired.

Parameters

pParams Callback parameters interface pointer.

Implements **PdkPlugin::P3dPluginCallback**.

◆ P3D::PdkPlugin::MessageCallback

class P3D::PdkPlugin::MessageCallback

Callback class that calls OnMessage

Inherits **PdkPlugin::P3dPluginCallback**.

Public Member Functions

MessageCallback (**PdkPlugin** ***pPlugin**)

virtual void Invoke (**IParameterListV400** ***pParams**) [override](#)

▶ Public Member Functions inherited from **PdkPlugin::P3dPluginCallback**

Additional Inherited Members

▶ Public Attributes inherited from **PdkPlugin::P3dPluginCallback**

Constructor & Destructor Documentation

◆ MessageCallback()

MessageCallback (**PdkPlugin** * **pPlugin**) [inline](#)

Member Function Documentation

◆ Invoke()

```
virtual void Invoke ( IParameterListV400 * pParams ) [inline] [override] [virtual]
```

Called when the registered event is fired.

Parameters

pParams Callback parameters interface pointer.

Implements **PdkPlugin::P3dPluginCallback**.

◆ P3D::PdkPlugin

class P3D::PdkPlugin

Base plugin implementation. This class registers for common Prepar3D events and messages, and provides easy access via a set of virtual functions.

Public Member Functions

PdkPlugin () noexcept

virtual ~**PdkPlugin ()**

virtual void **OnOneHz (IParameterListV400 *pParams)**

virtual void **OnFrame (IParameterListV400 *pParams)**

virtual void **OnLoadComplete (IParameterListV400 *pParams)**

virtual void **OnPluginsLoaded (IParameterListV400 *pParams)**

virtual void **OnViewChanged (IParameterListV400 *pParams)**

virtual void **OnViewCreated (IParameterListV400 *pParams)**

virtual void **OnViewDestroyed (IParameterListV400 *pParams)**

virtual void **OnVehicleChanged (IParameterListV400 *pParams)**

virtual void **OnCustomRender (IParameterListV400 *pParams)**

virtual void **OnApplicationShutdown (IParameterListV400 *pParams)**

virtual void **OnMessage (IParameterListV400 *pParams)**

virtual void **RegisterEvents ()**

virtual void **UnregisterEvents ()**

Protected Attributes

CComPtr< P3dPluginCallback > **m_spCustomRenderCallback**

CComPtr< P3dPluginCallback > **m_spOneHzCallback**

CComPtr< P3dPluginCallback > **m_spMessageCallback**

CComPtr< P3dPluginCallback > **m_spFrameCallback**

Constructor & Destructor Documentation

◆ PdkPlugin()

PdkPlugin() [inline](#) [noexcept](#)

◆ ~PdkPlugin()

virtual ~PdkPlugin() [inline](#) [virtual](#)

Member Function Documentation

◆ OnApplicationShutdown()

virtual void OnApplicationShutdown (**IParameterListV400** * pParams) [inline](#) [virtual](#)

Called when application shutdown has been requested. No pdk services or plugins have shut down yet, so it should be safe to use all services to unregister for callbacks and clean up local resources.

◆ OnCustomRender()

virtual void OnCustomRender (**IParameterListV400** * pParams) [inline](#) [virtual](#)

Called every frame for each 3d view. At this point, Prepar3D is still building the 3d scene, and it is safe to add new objects using the IObjectRenderer service

◆ OnFrame()

virtual void OnFrame (**IParameterListV400** * pParams) [inline](#) [virtual](#)

Called once per frame

◆ OnLoadComplete()

virtual void OnLoadComplete (**IParameterListV400** * pParams) [inline](#) [virtual](#)

Called when loading of a scenario is complete

◆ OnMessage()

virtual void OnMessage (**IParameterListV400** * pParams) [inline](#) [virtual](#)

Called when a message event is fired. This function calls OnLoadComplete, OnPluginsLoaded, OnApplicationShutdown, OnViewChanged, OnViewCreated, OnViewDestroyed, and

OnVehicleChanged. Any overrides of this function should call back to the base to ensure the callbacks continue to function.

◆ OnOneHz()

virtual void OnOneHz(**IParameterListV400** * pParams) **inline** **virtual**

Called once per second

◆ OnPluginsLoaded()

virtual void OnPluginsLoaded(**IParameterListV400** * pParams) **inline** **virtual**

Called after all dll-based plugins have loaded. This is a good place to query for pdk services registered by other plugins.

◆ OnVehicleChanged()

virtual void OnVehicleChanged(**IParameterListV400** * pParams) **inline** **virtual**

Called whenever the user vehicle is changed

◆ OnViewChanged()

virtual void OnViewChanged(**IParameterListV400** * pParams) **inline** **virtual**

Called whenever a window changes views or sets a new camera definition

◆ OnViewCreated()

virtual void OnViewCreated(**IParameterListV400** * pParams) **inline** **virtual**

Called whenever a view is created

◆ OnViewDestroyed()

virtual void OnViewDestroyed(**IParameterListV400** * pParams) **inline** **virtual**

Called whenever a view is destroyed

◆ RegisterEvents()

virtual void RegisterEvents() inline virtual

Register event callbacks for this plugin.

◆ **UnregisterEvents()**

virtual void UnregisterEvents() inline virtual

Unegister event callbacks for this plugin.

Member Data Documentation

◆ **m_spCustomRenderCallback**

CComPtr<P3dPluginCallback> m_spCustomRenderCallback protected

◆ **m_spFrameCallback**

CComPtr<P3dPluginCallback> m_spFrameCallback protected

◆ **m_spMessageCallback**

CComPtr<P3dPluginCallback> m_spMessageCallback protected

◆ **m_spOneHzCallback**

CComPtr<P3dPluginCallback> m_spOneHzCallback protected

◆ **P3D::PdkServices**

class P3D::PdkServices

Provides static access to all of Prepar3D's core PDK services.

Remarks

To use this in a plugin:

- Include **initpdk.h** from the cpp file where dll start and stop functions are defined
- Call **Init()** in the dll start function
- Call **Shutdown()** from the dll stop function

```
#include "initpdk.h"

void DLLStart(__in __notnull IPdk* pPdk)
{
    PdkServices::Init(pPdk);
    // init code for your plugin
```

```

}

void DLLStop(void)
{
    // deinit code for your plugin
    PdkServices::Shutdown();
}

```

Public Member Functions

PdkServices (IPdk *pPdk)

~PdkServices ()

Static Public Member Functions

static void Init (IPdk *pPdk)

static void Shutdown ()

static IPdk * GetPdk ()

Get Pdk. [More...](#)

static IEventServiceV400 * GetEventService ()

Get Event Service. [More...](#)

static IVisualEffectManagerV430 * GetVisualEffectManager ()

Get Visual Effect Manager. [More...](#)

static IDataLoadHelperV400 * GetDataLoadHelper ()

Get Data Load Helper. [More...](#)

static IRenderingPluginSystemV500 * GetRenderingPluginSystem ()

Get Rendering Plugin System. [More...](#)

static IWindowPluginSystemV440 * GetWindowPluginSystem ()

Get Window Plugin System. [More...](#)

static IGlobalDataV430 * GetGlobalData ()

Get Global Data. [More...](#)

static IObjectRendererV500 * GetObjectRenderer ()

Get Object Renderer. [More...](#)

static IWeatherSystemV500 * GetWeatherSystem ()

Get Weather System. [More...](#)

static ISimObjectManagerV500 * GetSimObjectManager ()

Get Simulation Object Manager. [More...](#)

static IReportingServiceV400 * GetReportingService ()

Get Reporting Service. [More...](#)

static IPanelSystemV400 * GetPanelSystem ()

Get Panel System. [More...](#)

static IIconServiceV410 * GetIconService ()

Get Icon Service. [More...](#)

static IMenuServiceV410 * GetMenuService ()

Get Menu Service. [More...](#)

static IMultiplayerServiceV453 * GetMultiplayerService ()

Get Multiplaver Service. [More...](#)

static IMultichannelServiceV440 *	GetMultichannelService () Get Multichannel Service. More...
static ICigiServiceV430 *	GetCigiService () Get CIGI Service. More...
static IConfigurationServiceV440 *	GetConfigurationService () Get Configuration Service. More...
static ISimPropertyServiceV440 *	GetSimPropertyService () Get Sim Property Service. More...
static IControllableCameraV450 *	GetControllableCamera () Get Controllable Camera. More...
static IVRServiceV452 *	GetVRService () Get VR Service. More...
static IWorldObjectServiceV450 *	GetWorldObjectService () Get World object Service. More...

Protected Attributes

CComPtr< IPdk >	m_spPdk
CComPtr< IEventServiceV400 >	m_spEventService
CComPtr< IVisualEffectManagerV430 >	m_spVisualEffectManager
CComPtr< IDataLoadHelperV400 >	m_spDataLoadHelper
CComPtr< IRenderingPluginSystemV500 >	m_spRenderingPluginSystem
CComPtr< IWindowPluginSystemV440 >	m_spWindowPluginSystem
CComPtr< IGlobalDataV430 >	m_spGlobalData
CComPtr< IObjectRendererV500 >	m_spObjectRenderer
CComPtr< IWeatherSystemV500 >	m_spWeatherSystem
CComPtr< ISimObjectManagerV500 >	m_spSimObjectManager
CComPtr< IReportingServiceV400 >	m_spReportingService
CComPtr< IPanelSystemV400 >	m_spPanelSystem
CComPtr< IconServiceV410 >	m_spIconService
CComPtr< IMenuServiceV410 >	m_spMenuService
CComPtr< IMultiplayerServiceV453 >	m_spMultiplayerService
CComPtr< IMultichannelServiceV440 >	m_spMultichannelService
CComPtr< ICigiServiceV430 >	m_spCigiService
CComPtr< IConfigurationServiceV440 >	m_spConfigurationService
CComPtr< ISimPropertyServiceV440 >	m_spSimPropertyService
CComPtr< IControllableCameraV450 >	m_spControllableCamera
CComPtr< IVRServiceV452 >	m_spVRService
CComPtr< IWorldObjectServiceV450 >	m_spWorldObjectService

Static Protected Attributes

static **PdkServices** * **m_pServices** = nullptr

Constructor & Destructor Documentation

◆ PdkServices()

PdkServices (**IPdk** * pPdk) **inline**

◆ ~PdkServices()

~PdkServices () **inline**

Member Function Documentation

◆ GetCigiService()

ICigiServiceV430 * GetCigiService () **static**

Get CIGI Service.

◆ GetConfigurationService()

IConfigurationServiceV440 * GetConfigurationService () **static**

Get Configuration Service.

◆ GetControllableCamera()

IControllableCameraV450 * GetControllableCamera () **static**

Get Controllable Camera.

◆ GetDataLoadHelper()

IDataLoadHelperV400 * GetDataLoadHelper () **static**

Get Data Load Helper.

◆ GetEventService()

IEventServiceV400 * GetEventService () **static**

Get Event Service.

◆ GetGlobalData()

IGlobalDataV430 * GetGlobalData() static

Get Global Data.

◆ GetIconService()

IIconServiceV410 * GetIconService() static

Get Icon Service.

◆ GetMenuService()

IMenuServiceV410 * GetMenuService() static

Get Menu Service.

◆ GetMultichannelService()

IMultichannelServiceV440 * GetMultichannelService() static

Get Multichannel Service.

◆ GetMultiplayerService()

IMultiplayerServiceV453 * GetMultiplayerService() static

Get Multiplayer Service.

◆ GetObjectRenderer()

IObjectRendererV500 * GetObjectRenderer() static

Get Object Renderer.

◆ GetPanelSystem()

IPanelSystemV400 * GetPanelSystem() static

Get Panel System.

◆ GetPdk()

IPdk * GetPdk() **static**

Get Pdk.

◆ GetRenderingPluginSystem()

IRenderingPluginSystemV500 * GetRenderingPluginSystem() **static**

Get Rendering Plugin System.

◆ GetReportingService()

IReportingServiceV400 * GetReportingService() **static**

Get Reporting Service.

◆ GetSimObjectManager()

ISimObjectManagerV500 * GetSimObjectManager() **static**

Get Simulation Object Manager.

◆ GetSimPropertyService()

ISimPropertyServiceV440 * GetSimPropertyService() **static**

Get Sim Property Service.

◆ GetVisualEffectManager()

IVisualEffectManagerV430 * GetVisualEffectManager() **static**

Get Visual Effect Manager.

◆ GetVRService()

IVRServiceV452 * GetVRService() **static**

Get VR Service.

◆ GetWeatherSystem()

IWeatherSystemV500 * GetWeatherSystem() **static**

Get Weather System.

◆ GetWindowPluginSystem()

IWindowPluginSystemV440 * GetWindowPluginSystem() **static**

Get Window Plugin System.

◆ GetWorldObjectService()

IWorldObjectServiceV450 * GetWorldObjectService() **static**

Get World object Service.

◆ Init()

void Init(**IPdk** * pPdk) **static**

Initialize static instance of **PdkServices**. This should be called from the dll start function.

◆ Shutdown()

void Shutdown() **static**

Shutdown static instance of **PdkServices**. This should be called from the dll stop function.

Member Data Documentation

◆ m_pServices

P3D::PdkServices * m_pServices = nullptr **static protected**

◆ m_spCigiService

CComPtr<ICigiServiceV430> m_spCigiService **protected**

◆ m_spConfigurationService

CComPtr<IConfigurationServiceV440> m_spConfigurationService protected

◆ m_spControllableCamera

CComPtr<IControllableCameraV450> m_spControllableCamera protected

◆ m_spDataLoadHelper

CComPtr<IDataLoadHelperV400> m_spDataLoadHelper protected

◆ m_spEventService

CComPtr<IEventServiceV400> m_spEventService protected

◆ m_spGlobalData

CComPtr<IGlobalDataV430> m_spGlobalData protected

◆ m_spIconService

CComPtr<IIIconServiceV410> m_spIconService protected

◆ m_spMenuService

CComPtr<IMenuServiceV410> m_spMenuService protected

◆ m_spMultichannelService

CComPtr<IMultichannelServiceV440> m_spMultichannelService protected

◆ m_spMultiplayerService

CComPtr<IMultiplayerServiceV453> m_spMultiplayerService protected

◆ m_spObjectRenderer

CComPtr<IOBJECTRENDERERV500> m_spObjectRenderer protected

◆ m_spPanelSystem

CComPtr<IPanelSystemV400> m_spPanelSystem protected

◆ m_spPdk

CComPtr<IPdk> m_spPdk protected

◆ m_spRenderingPluginSystem

CComPtr<IRenderingPluginSystemV500> m_spRenderingPluginSystem protected

◆ m_spReportingService

CComPtr<IReportingServiceV400> m_spReportingService protected

◆ m_spSimObjectManager

CComPtr<ISimObjectManagerV500> m_spSimObjectManager protected

◆ m_spSimPropertyService

CComPtr<ISimPropertyServiceV440> m_spSimPropertyService protected

◆ m_spVisualEffectManager

CComPtr<IVisualEffectManagerV430> m_spVisualEffectManager protected

◆ m_spVRService

CComPtr<IVRServiceV452> m_spVRService protected

◆ m_spWeatherSystem

CComPtr<IWeatherSystemV500> m_spWeatherSystem protected

◆ m_spWindowPluginSystem

CComPtr<IWindowPluginSystemV440> m_spWindowPluginSystem protected

◆ m_spWorldObjectService

CComPtr<IWorldObjectServiceV450> m_spWorldObjectService protected

Variables

REFIID IID_IPdk = __uuidof(IPdk)
REFIID IID_IPdkV01 = __uuidof(IPdkV01)

Variable Documentation

◆ IID_IPdk

REFIID IID_IPdk = __uuidof(IPdk)

◆ IID_IPdkV01

REFIID IID_IPdkV01 = __uuidof(IPdkV01)

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

PDK Types

Overview

The window and camera services enables external applications to control basic camera functions as well as adding/removing post-process effects and sensor modes using window plug-ins.

Classes

struct	LLADegreesMeters
struct	PBHDegrees
struct	XYZMeters
struct	ScreenCoord
struct	ObjectLocalTransform
struct	ObjectWorldTransform
union	ARGBColor
struct	ARGBColor.__unnamed__
union	RenderFlags
struct	RenderFlags.__unnamed__
struct	BoundingBox2D
struct	TextDescription
union	TextDescription.__unnamed__ Union that supports text description flags. More...
struct	TextDescription.__unnamed__.__unnamed__
class	BasicWaypoint BasicWaypoint . More...
class	EntityType
struct	CloudLayer
struct	WindAloftLayer
struct	TempLayer
struct	PressureInfo

struct **VisibilityLayer**
class **IListBuilder< T >**
class **CComPtrVecBuilder< T >**
class **VecListBuilder< T >**
class **NameListCopy**
class **NameList**
class **NameListC**

Class Documentation

◆ LLADegreesMeters

struct LLADegreesMeters

Struct that stores world position lat/lon in degrees, and altitude in meters.

Public Member Functions

LLADegreesMeters () noexcept

LLADegreesMeters (double latitude, double longitude, double altitude)

Public Attributes

double **Latitude**

double **Longitude**

double **Altitude**

Constructor & Destructor Documentation

◆ LLADegreesMeters() [1/2]

LLADegreesMeters () inline noexcept

◆ LLADegreesMeters() [2/2]

**LLADegreesMeters (double latitude,
 double longitude,
 double altitude
)** inline

Parameters

latitude Latitude in degrees

longitude longitude in degrees

altitude altitude in meters

Member Data Documentation

◆ Altitude

double Altitude

◆ Latitude

double Latitude

◆ Longitude

double Longitude

◆ PBHDegrees

struct PBHDegrees

Struct that stores orientation PBH (pitch, bank, heading) in degrees

Public Member Functions

PBHDegrees () noexcept

PBHDegrees (float pitch, float bank, float heading)

Public Attributes

float **Pitch**

float **Bank**

float **Heading**

Constructor & Destructor Documentation

◆ PBHDegrees() [1/2]

PBHDegrees () inline noexcept

◆ PBHDegrees() [2/2]

PBHDegrees (float pitch,
float bank,

```
    float heading  
)  
    inline
```

Parameters

pitch pitch in degrees
bank bank in degrees
heading heading in degrees

Member Data Documentation

◆ Bank

```
float Bank
```

◆ Heading

```
float Heading
```

◆ Pitch

```
float Pitch
```

◆ XYZMeters

```
struct XYZMeters
```

Public Member Functions

```
XYZMeters () noexcept  
XYZMeters (float x, float y, float z)  
XYZMeters operator - (const XYZMeters &rhs) const  
XYZMeters & operator -= (const XYZMeters &rhs)  
XYZMeters operator+ (const XYZMeters &rhs) const  
XYZMeters & operator+= (const XYZMeters &rhs)  
XYZMeters operator * (const XYZMeters &rhs) const  
XYZMeters & operator *= (const XYZMeters &rhs)  
XYZMeters operator * (const float &rhs) const  
XYZMeters & operator *= (const float &rhs)  
XYZMeters operator/ (const XYZMeters &rhs) const  
XYZMeters & operator/= (const XYZMeters &rhs)  
XYZMeters operator/ (const float &rhs) const  
XYZMeters & operator/= (const float &rhs)
```

XYZMeters & **operator** (const float &rhs)

```
bool operator==(const XYZMeters &src) const  
bool operator!=(const XYZMeters &src) const
```

Public Attributes

```
float X  
float Y  
float Z
```

Constructor & Destructor Documentation

◆ XYZMeters() [1/2]

XYZMeters() **inline** **noexcept**

◆ XYZMeters() [2/2]

XYZMeters(float x,
 float y,
 float z
)
 inline

Parameters

x x offset in meters
y y offset in meters
z z offset in meters

Member Function Documentation

◆ operator !=()

bool operator !=(const XYZMeters & src) const **inline**

◆ operator *() [1/2]

XYZMeters operator *(const XYZMeters & rhs) const **inline**

◆ operator *() [2/2]

XYZMeters operator *(const float & rhs) const **inline**

◆ operator *=() [1/2]

XYZMeters& operator *= (const **XYZMeters** & **rhs**) **inline**

◆ operator *=() [2/2]

XYZMeters& operator *= (const float & **rhs**) **inline**

◆ operator -()

XYZMeters operator - (const **XYZMeters** & **rhs**) const **inline**

◆ operator -=()

XYZMeters& operator -= (const **XYZMeters** & **rhs**) **inline**

◆ operator+()

XYZMeters operator+ (const **XYZMeters** & **rhs**) const **inline**

◆ operator+=(())

XYZMeters& operator+= (const **XYZMeters** & **rhs**) **inline**

◆ operator/() [1/2]

XYZMeters operator/ (const **XYZMeters** & **rhs**) const **inline**

◆ operator/() [2/2]

XYZMeters operator/ (const float & **rhs**) const **inline**

◆ operator/=(()) [1/2]

XYZMeters& operator/= (const **XYZMeters** & **rhs**) **inline**

◆ operator/=(()) [2/2]

XYZMeters& operator/= (const float & **rhs**) **inline**

◆ operator==(())

bool operator== (const **XYZMeters** & **src**) const **inline**

Member Data Documentation

◆ X

float X

◆ Y

float Y

◆ Z

float Z

◆ ScreenCoord

struct ScreenCoord

Public Member Functions

ScreenCoord () noexcept

ScreenCoord (float x, float y, float dist)

Public Attributes

float **XPixels**

float **YPixels**

float **DistanceMeters**

Constructor & Destructor Documentation

◆ **ScreenCoord() [1/2]**

ScreenCoord () **inline** **noexcept**

◆ **ScreenCoord() [2/2]**

**ScreenCoord (float x,
 float y,
 float dist
)** **inline**

Parameters

x x offset in meters
y y offset in meters
z z offset in meters

Member Data Documentation

◆ DistanceMeters

float DistanceMeters

◆ XPixels

float XPixels

◆ YPixels

float YPixels

◆ ObjectLocalTransform

struct ObjectLocalTransform

Defines a local transformation which includes a position and orientation offset

Public Member Functions

[ObjectLocalTransform \(\) noexcept](#)

[ObjectLocalTransform \(float x, float y, float z, float pitch, float bank, float heading\)](#)

Public Attributes

[XYZMeters XYZ](#)

[PBHDegrees PBH](#)

Constructor & Destructor Documentation

◆ ObjectLocalTransform() [1/2]

[ObjectLocalTransform \(\) \[inline\] noexcept](#)

◆ ObjectLocalTransform() [2/2]

```
ObjectLocalTransform ( float x,  
                      float y,  
                      float z,  
                      float pitch,  
                      float bank,  
                      float heading  
                    )
```

inline

Parameters

x x offset in meters
y y offset in meters
z z offset in meters
pitch pitch in degrees
bank bank in degrees
heading heading in degrees

Member Data Documentation

◆ PBH

PBHDegrees PBH

◆ XYZ

XYZMeters XYZ

◆ ObjectWorldTransform

struct ObjectWorldTransform

Defines a world transformation which includes an LLA for position and PBH for orientation. Altitude is in meters. All angle values are in degrees.

Public Member Functions

ObjectWorldTransform () noexcept

ObjectWorldTransform (double latitude, double longitude, double altitude, float pitch, float bank, float heading)

Public Attributes

LLADegreesMeters LLA

PBHDegrees PBH

Constructor & Destructor Documentation

◆ ObjectWorldTransform() [1/2]

ObjectWorldTransform () **inline** **noexcept**

◆ ObjectWorldTransform() [2/2]

ObjectWorldTransform (double **latitude**,
 double **longitude**,
 double **altitude**,
 float **pitch**,
 float **bank**,
 float **heading**
)

inline

Parameters

latitude Latitude in degrees
longitude longitude in degrees
altitude altitude in meters
pitch pitch in degrees
bank bank in degrees
heading heading in degrees

Member Data Documentation

◆ LLA

LLADegreesMeters LLA

◆ PBH

PBHDegrees PBH

◆ ARGBColor

union ARGBColor

Union that stores color in ARGB 8 bit format

Public Member Functions

ARGBColor () **noexcept**

ARGBColor (unsigned int alpha, unsigned int red, unsigned int green, unsigned int blue)

Public Attributes

```
unsigned int Color
```

```
struct {
```

```
    unsigned int Blue: 8
```

```
    unsigned int Green: 8
```

```
    unsigned int Red: 8
```

```
    unsigned int Alpha: 8
```

```
};
```

Constructor & Destructor Documentation

◆ ARGBColor() [1/2]

```
ARGBColor( ) inline noexcept
```

◆ ARGBColor() [2/2]

```
ARGBColor( unsigned int alpha,  
            unsigned int red,  
            unsigned int green,  
            unsigned int blue  
        ) inline
```

Parameters

alpha Alpha component of the color

red Red component of the color

green Green component of the color

blue Blue component of the color

Member Data Documentation

◆ @51

```
struct { ... }
```

◆ Color

```
unsigned int Color
```

◆ ARGBColor.__unnamed__

```
struct ARGBColor.__unnamed__
```

Class Members

unsigned int	Alpha: 8
unsigned int	Blue: 8
unsigned int	Green: 8
unsigned int	Red: 8

◆ RenderFlags

union RenderFlags

Union that stores **RenderFlags** to control drawing

Public Member Functions

RenderFlags () noexcept

RenderFlags (unsigned int flags)

Public Attributes

unsigned int **Flags**

```
struct {
    bool DrawFromBase: 1
    bool DrawWithVC: 1
    bool DepthReadDisable: 1
    bool AlphaWriteEnable: 1
    bool ActAsStencil: 1
};
```

Constructor & Destructor Documentation

◆ RenderFlags() [1/2]

RenderFlags () **inline** **noexcept**

◆ RenderFlags() [2/2]

RenderFlags (unsigned int flags) **inline**

Member Data Documentation

◆ @53

struct { ... }

◆ Flags

unsigned int Flags

◆ RenderFlags.__unnamed__

struct RenderFlags.__unnamed__

Class Members

bool	ActAsStencil: 1	
bool	AlphaWriteEnable: 1	
bool	DepthReadDisable: 1	
bool	DrawFromBase: 1	
bool	DrawWithVC: 1	

◆ BoundingBox2D

struct BoundingBox2D

Describes a 2D bounding box in pixels.

Public Member Functions

[BoundingBox2D \(\)](#)

Public Attributes

int **left**

int **top**

int **right**

int **bottom**

Constructor & Destructor Documentation

◆ BoundingBox2D()

[BoundingBox2D \(\)](#) [inline](#)

Member Data Documentation

◆ bottom

int bottom

◆ left

int left

◆ right

int right

◆ top

int top

◆ TextDescription

struct TextDescription

Describes how 2D and 3D text should be drawn to the screen.

Public Member Functions

[TextDescription \(\)](#)

Public Attributes

TEXT_FONT Font

HORIZONTAL_ALIGNMENT HorizontalAlignment

VERTICAL_ALIGNMENT VerticalAlignment

BoundingBox2D BoundingBox

union {

struct {

bool WorldSpace: 1

bool DisplayOnTop: 1

bool DropShadow: 1

bool CalculateBox: 1

bool NoPostProcess: 1

}

unsigned int Flags

};

Union that supports text description flags. More...

Constructor & Destructor Documentation

◆ **TextDescription()**

TextDescription() `inline`

Member Data Documentation

◆ **@55**

`union { ... }`

Union that supports text description flags.

◆ **BoundingBox**

BoundingBox2D `BoundingBox`

The size of the text's bounding box.

Remarks

Negative values are currently not supported.

◆ **Font**

TEXT_FONT `Font`

The font type.

◆ **HorizontalAlignment**

HORIZONTAL_ALIGNMENT `HorizontalAlignment`

Alignment in the horizontal direction.

◆ **VerticalAlignment**

VERTICAL_ALIGNMENT `VerticalAlignment`

Alignment in the vertical direction.

◆ **TextDescription.__unnamed__**

`union TextDescription.__unnamed__`

Union that supports text description flags.

Class Members

<u>unnamed</u>	<u>unnamed</u>	
unsigned int	Flags	

◆ `TextDescription.__unnamed__.__unnamed__`

struct `TextDescription.__unnamed__.__unnamed__`

Class Members

bool	CalculateBox: 1	True if the bounding box should be calculated based on the given string, false if the user-defined bounding box should be used.
bool	DisplayOnTop: 1	True if text is to be rendered on top of the scene.
bool	DropShadow: 1	True if a drop shadow should be drawn with the text.
bool	NoPostProcess: 1	Prevents post-process from being applied to the text.
bool	WorldSpace: 1	True if the text box should be draw in world space, false if it should be drawn in screen space.

◆ `P3D::BasicWaypoint`

class `P3D::BasicWaypoint`

BasicWaypoint.

Class Members

double	dHeading	
DXYZ	vLonAltLat	

◆ `P3D::EntityType`

class `P3D::EntityType`

Public Member Functions

EntityType () noexcept

unsigned char **GetKind () const**

void **SetKind (unsigned char byKind)**

unsigned char **GetDomain () const**

void **SetDomain (unsigned char byDomain)**

unsigned short **GetCountry () const**

void **SetCountry (unsigned short usCountry)**

unsigned char **GetCategory () const**

void **SetCategory (unsigned char byCategory)**

unsigned char **GetSubcategory () const**

```
unsigned char GetSubcategory() const  
void SetSubcategory(unsigned char bySubcategory)  
unsigned char GetSpecific() const  
void SetSpecific(unsigned char bySpecific)  
unsigned char GetExtra() const  
void SetExtra(unsigned char byExtra)
```

Private Attributes

```
unsigned char m_byKind  
unsigned char m_byDomain  
unsigned short m_usCountry  
unsigned char m_byCategory  
unsigned char m_bySubcategory  
unsigned char m_bySpecific  
unsigned char m_byExtra
```

Constructor & Destructor Documentation

◆ EntityType()

```
EntityType( ) inline noexcept
```

Member Function Documentation

◆ GetCategory()

```
unsigned char GetCategory( ) const inline
```

Gets DIS entity category

◆ GetCountry()

```
unsigned short GetCountry( ) const inline
```

Gets DIS entity country

◆ GetDomain()

```
unsigned char GetDomain( ) const inline
```

Gets DIS entity domain

◆ **GetExtra()**

unsigned char GetExtra() const [inline](#)

Gets DIS entity extra

◆ **GetKind()**

unsigned char GetKind() const [inline](#)

Gets DIS entity kind

◆ **GetSpecific()**

unsigned char GetSpecific() const [inline](#)

Gets DIS entity specific

◆ **GetSubcategory()**

unsigned char GetSubcategory() const [inline](#)

Gets DIS entity subcategory

◆ **SetCategory()**

void SetCategory(unsigned char [byCategory](#)) [inline](#)

Sets DIS entity category

◆ **SetCountry()**

void SetCountry(unsigned short [usCountry](#)) [inline](#)

Sets DIS entity country

◆ **SetDomain()**

void SetDomain(unsigned char [byDomain](#)) [inline](#)

Sets DIS entity domain

◆ SetExtra()

void SetExtra (unsigned char byExtra) inline

Sets DIS entity extra

◆ SetKind()

void SetKind (unsigned char byKind) inline

Sets DIS entity kind

◆ SetSpecific()

void SetSpecific (unsigned char bySpecific) inline

Sets DIS entity specific

◆ SetSubcategory()

void SetSubcategory (unsigned char bySubcategory) inline

Sets DIS entity subcategory

Member Data Documentation

◆ m_byCategory

unsigned char m_byCategory private

DIS entity category

◆ m_byDomain

unsigned char m_byDomain private

DIS entity domain

◆ m_byExtra

unsigned char m_byExtra private

DIS entity extra

◆ m_byKind

unsigned char m_byKind private

DIS entity kind

◆ m_bySpecific

unsigned char m_bySpecific private

DIS entity specific

◆ m_bySubcategory

unsigned char m_bySubcategory private

DIS entity subcategory

◆ m_usCountry

unsigned short m_usCountry private

DIS entity country

◆ P3D::CloudLayer

struct P3D::CloudLayer

Class Members

CLOUD_COVER	eCloudCover	
CLOUD_TOP	eCloudTop	
CLOUD_TYPE	eCloudType	
ICINGRATE	elcingRate	
PRECIPRATE	ePrecipRate	
PRECIPTYPE	ePrecipType	
TURBULANCE	eTurbulence	
float	fCloudBase	
float	fCloudDeviation	
float	fCloudTops	
float	fPrecipBase	

◆ P3D::WindAloftLayer

struct P3D::WindAloftLayer

Class Members

TURBULANCE	eTurb
WINDSHEAR	eWindShear
float	fAlt
float	fDirection
float	fGusts
float	fSpeed
float	fVariance

◆ P3D::TempLayer

struct P3D::TempLayer

Class Members

float	fAlt
float	fDewPoint
float	fRange
float	fTemp

◆ P3D::PressureInfo

struct P3D::PressureInfo

Class Members

float	fPressureAtAlt
float	fPressureAtSL
float	fRange

◆ P3D::VisibilityLayer

struct P3D::VisibilityLayer

Class Members

float	fBase
float	fTops
float	fVis

◆ P3D::IListBuilder

class P3D::IListBuilder

```
template<class T>
class P3D::IListBuilder< T >
```

Templated List Building Interface used by Prepar3D to add requested items to a list.

Inherited by [CComPtrVecBuilder< T >](#), and [VecListBuilder< T >](#).

Public Member Functions

```
virtual bool AddItem (T *item) override
virtual void BeginBuilding () override
virtual void EndBuilding ()
```

Member Function Documentation

◆ AddItem()

```
virtual bool AddItem ( T * item ) pure virtual
```

Called for each item until all items have been called or until AddItem returns false.

Returns

true to continue adding items or false to stop

Implemented in [NameListC](#), [NameList](#), [NameListCopy](#), [VecListBuilder< T >](#), and [CComPtrVecBuilder< T >](#).

◆ BeginBuilding()

```
virtual void BeginBuilding ( ) pure virtual
```

Called before any items are added.

Implemented in [NameListC](#), [NameList](#), [NameListCopy](#), [VecListBuilder< T >](#), and [CComPtrVecBuilder< T >](#).

◆ EndBuilding()

```
virtual void EndBuilding ( ) inline virtual
```

Called after all items have been added.

◆ P3D::CComPtrVecBuilder

```
class P3D::CComPtrVecBuilder
```

```
template<class T>
```

class P3D::CComPtrVecBuilder< T >

List builder implementation that creates a standard vector of CComPtrs of a templated type.

Inherits [IListBuilder< T >](#).

Public Member Functions

virtual bool [AddItem](#) (T *item) override

virtual void [BeginBuilding](#) () override

► [Public Member Functions inherited from IListBuilder< T >](#)

Public Attributes

std::vector< CComPtr< T > > [Items](#)

Member Function Documentation

◆ AddItem()

virtual bool AddItem (T * item) [inline](#) [override](#) [virtual](#)

Called for each item until all items have been called or until AddItem returns false.

Returns

true to continue adding items or false to stop

Implements [IListBuilder< T >](#).

◆ BeginBuilding()

virtual void BeginBuilding () [inline](#) [override](#) [virtual](#)

Called before any items are added.

Implements [IListBuilder< T >](#).

Member Data Documentation

◆ Items

std::vector<CComPtr<T> > Items

◆ P3D::VecListBuilder

class P3D::VecListBuilder

```
template<class T>
class P3D::VecListBuilder< T >
```

List builder implementation that creates a standard vector of objects of a templated type.

Inherits [IListBuilder< T >](#).

Public Member Functions

virtual bool [AddItem](#) (T *item) override

virtual void [BeginBuilding](#) () override

▶ [Public Member Functions inherited from IListBuilder< T >](#)

Public Attributes

std::vector< T > [Items](#)

Member Function Documentation

◆ AddItem()

virtual bool AddItem (T * item) [inline](#) [override](#) [virtual](#)

Called for each item until all items have been called or until AddItem returns false.

Returns

true to continue adding items or false to stop

Implements [IListBuilder< T >](#).

◆ BeginBuilding()

virtual void BeginBuilding () [inline](#) [override](#) [virtual](#)

Called before any items are added.

Implements [IListBuilder< T >](#).

Member Data Documentation

◆ Items

std::vector<T> Items

◆ P3D::NameListCopy

class P3D::NameListCopy

IListBuilder that stores names in a string vector. The contents of each name are copied into the string so it is safe to store this list for later use.

Inherits **IListBuilder< const WCHAR >**.

Public Member Functions

virtual bool **AddItem** (const WCHAR *item) override
virtual void **BeginBuilding** () override

► Public Member Functions inherited from **IListBuilder< const WCHAR >**

Public Attributes

std::vector< std::basic_string< WCHAR > > **Items**

Member Function Documentation

◆ AddItem()

virtual bool AddItem (const WCHAR * item) **inline** **override** **virtual**

Called for each item until all items have been called or until AddItem returns false.

Returns

true to continue adding items or false to stop

Implements **IListBuilder< const WCHAR >**.

◆ BeginBuilding()

virtual void BeginBuilding () **inline** **override** **virtual**

Called before any items are added.

Implements **IListBuilder< const WCHAR >**.

Member Data Documentation

◆ Items

std::vector<std::basic_string<WCHAR> > **Items**

Vector of strings that can be used to access the name list once it has been built.

◆ P3D::NameList

```
class P3D::NameList
```

IListBuilder that stores names in a temporary const char* vector. The contents of each string are not copied, so this type is only safe to use within the scope of the current function.

Inherits **IListBuilder< const WCHAR >**.

Public Member Functions

virtual bool **AddItem** (const WCHAR *item) override

virtual void **BeginBuilding** () override

▶ Public Member Functions inherited from **IListBuilder< const WCHAR >**

Public Attributes

std::vector< const WCHAR * > **Items**

Member Function Documentation

◆ AddItem()

virtual bool AddItem (const WCHAR * item) **inline** **override** **virtual**

Called for each item until all items have been called or until AddItem returns false.

Returns

true to continue adding items or false to stop

Implements **IListBuilder< const WCHAR >**.

◆ BeginBuilding()

virtual void BeginBuilding () **inline** **override** **virtual**

Called before any items are added.

Implements **IListBuilder< const WCHAR >**.

Member Data Documentation

◆ Items

std::vector<const WCHAR*> **Items**

const char* vector that can be used to access the name list once it has been built.

◆ P3D::NameListC

class P3D::NameListC

IListBuilder that stores names in a preallocated fixed size array.

Inherits **IListBuilder< const WCHAR >**.

Public Member Functions

NameListC (const WCHAR **names, int &count, bool bMakeCopy=false)

virtual bool **AddItem** (const WCHAR *item) override

virtual void **BeginBuilding** () override

▶ Public Member Functions inherited from **IListBuilder< const WCHAR >**

Private Attributes

const WCHAR ** **m_aNames**

int **m_iMaxCount**

int & **m_iCount**

bool **m_bMakeCopy**

Constructor & Destructor Documentation

◆ NameListC()

NameListC (const WCHAR ** **names**,
 int & **count**,
 bool **bMakeCopy** = **false**
)

inline

Constructor which takes preallocated fixed size array to hold this lists data.

Parameters

names pointer to preallocated const char* array

count size of preallocated array. Reference value that will be changed to hold the count of items added to the list.

Member Function Documentation

◆ AddItem()

virtual bool **AddItem** (const WCHAR * **item**) **inline** **override** **virtual**

Called for each item until all items have been called or until AddItem returns false.

Returns

true to continue adding items or false to stop

Implements **IListBuilder< const WCHAR >**.

◆ **BeginBuilding()**

virtual void BeginBuilding () **inline** **override** **virtual**

Called before any items are added.

Implements **IListBuilder< const WCHAR >**.

Member Data Documentation

◆ **m_aNames**

const WCHAR** m_aNames **private**

◆ **m_bMakeCopy**

bool m_bMakeCopy **private**

◆ **m_iCount**

int& m_iCount **private**

◆ **m_iMaxCount**

int m_iMaxCount **private**

Macros

#define **DXYZ** P3DDXYZ

Simple vector structures for passing between **P3D** host and implementation. More...

#define **FXYZ** P3DFXYZ

#define **DEFAULT_REFCOUNT_INLINE_IMPL()**

#define **DECLARE_IUNKNOWN_WITH_INLINE_REFCOUNT_IMPL()**

#define **NO_COPY_IUNKNOWN_IMPL**(IClassName)

Typedefs

typedef HRESULT(STDMETHODCALLTYPE * **PSimCreateFunc**) (**_in __notnull**
IBaseObjectV400 *, **_out __notnull** **ISimObject**
******)

typedef HRESULT(STDMETHODCALLTYPE * **PSaveLoadCallback**) (**_in** **LPCWSTR**

	pszSection, __in unsigned int ulInstance, __in LPCWSTR pszKeyword, __inout void *pvVal, __in const SAVED_DATA_TYPE eDataType)
typedef void(STDMETHODCALLTYPE *	PNewScenarioNotify) (BOOL bOnSave)
typedef HRESULT(STDMETHODCALLTYPE *	PPropertyCallback) (__in const ISimObject &Sim, __out double &dProperty, __in int iIndex) Double property callback. More...
typedef HRESULT(STDMETHODCALLTYPE *	PPropertyVectorCallback) (__in const ISimObject &Sim, __out DXYZ &vProperty, __in int iIndex) Vector (double x,y,z) property callback. More...
typedef HRESULT(STDMETHODCALLTYPE *	PPropertyStringCallback) (__in const ISimObject &Sim, __out LPWSTR pszProperty, __in UINT uStringLength, __in int iIndex) String property callback. More...
typedef HRESULT(STDMETHODCALLTYPE *	PPropertyCallbackWithSubString) (__in const ISimObject &Sim, __out double &dProperty, __in LPCWSTR pszSecondarySubstring, __in int iIndex) Double property callback (with secondary string input) More...
typedef HRESULT(STDMETHODCALLTYPE *	PEventCallback) (__in ISimObject &Sim, __in double dProperty, __in int iIndex) Event input callback. More...
typedef HRESULT(STDMETHODCALLTYPE *	PEventVectorCallback) (__in ISimObject &Sim, __in const DXYZ &vProperty, __in int iIndex) Event input callback. More...
typedef HRESULT(STDMETHODCALLTYPE *	PEventStringCallback) (__in ISimObject &Sim, __in LPCWSTR pszProperty, __in int iIndex) Event input callback. More...
typedef HRESULT(STDMETHODCALLTYPE *	PEventCallbackWithSubString) (__in ISimObject &Sim, __in double dProperty, __in LPCWSTR pszSecondarySecondarySubstring, __in int iIndex) Event input callback for double (with secondary string input) More...
typedef HRESULT(STDMETHODCALLTYPE *	POnObjectCreateCallback) (__in IUnknown &Obj) Register callback on object creation. This could be a simobject or a library object. More...
typedef HRESULT(STDMETHODCALLTYPE *	POnObjectRemoveCallback) (__in IUnknown &Obj) Register callback on object removal. This could be a simobject or a library object. More...
typedef HRESULT(STDMETHODCALLTYPE *	POnUserObjectChangedCallback) (__in IUnknown &NewObj, __in IUnknown &OldObj) Register callback on user object changing from one user to another (during runtime, not scenario load or vehicle change via UI) More...

using **INameList = IIlistBuilder< const WCHAR >**

Enumerations

enum	DYNAMIC_LIGHT { DYNAMIC_LIGHT_POINT , DYNAMIC_LIGHT_SPOT }
	Dynamic light type enum. More...
enum	TEXT_FONT { TEXT_FONT_DEFAULT , TEXT_FONT_SMALL , TEXT_FONT_SMALL_BOLD , TEXT_FONT_MEDIUM_BOLD , TEXT_FONT_LARGE_BOLD , TEXT_FONT_XLARGE_BOLD }
	Text font type enum. More...
enum	HORIZONTAL_ALIGNMENT { HORIZONTAL_ALIGNMENT_LEFT , HORIZONTAL_ALIGNMENT_CENTER , HORIZONTAL_ALIGNMENT_RIGHT }
	Text horizontal alignment enum. More...
enum	VERTICAL_ALIGNMENT { VERTICAL_ALIGNMENT_TOP , VERTICAL_ALIGNMENT_CENTER , VERTICAL_ALIGNMENT_BOTTOM }
	Text vertical alignment enum. More...
enum	COLLISIONTYPE { COLLISIONTYPE_NONE , COLLISIONTYPE_SIMOBJECT , COLLISIONTYPE_BUILDING , COLLISIONTYPE_TERRAIN , COLLISIONTYPE_OTHER }
enum	EVENTTYPE { EVENTTYPE_NORMAL , EVENTTYPE_AXIS , EVENTTYPE_POV , EVENTTYPE_NORMAL_NATIVE_OVERRIDE , EVENTTYPE_AXIS_NATIVE_OVERRIDE , EVENTTYPE_POV_NATIVE_OVERRIDE , EVENTTYPE_NOT_MAPPABLE }
	Type of keyboard or joystick mapping associated with an event being registered. More...
enum	PROPERTY_TYPE { PROPERTY_TYPE_EVENT , PROPERTY_TYPE_EVENT_WITH_SUBSTRING_INPUT , PROPERTY_TYPE_EVENT_VECTOR , PROPERTY_TYPE_EVENT_STRING , PROPERTY_TYPE_DOUBLE , PROPERTY_TYPE_DOUBLE_WITH_SUBSTRING_INPUT , PROPERTY_TYPE_STRING , PROPERTY_TYPE_VECTOR , NUM_PROPERTY_TYPES }
	Types of properties. More...
enum	SAVED_DATA_TYPE { SAVED_DATA_TYPE_DOUBLE , SAVED_DATA_TYPE_STRING , NUM_SAVED_DATA_TYPES }
	Types of data saved via the PSaveLoadCallback. More...
enum	NET_MODE_TYPE { NET_MODE_TYPE_NORMAL , NET_MODE_TYPE_REMOTE , NET_MODE_TYPE_MASTER , NET_MODE_TYPE_SLAVE }
	Types of network modes for a given object. More...
enum	INTERROGATIONTYPE { INTERROGATIONTYPE_NONE = 0, INTERROGATIONTYPE_TERRAIN = 1, INTERROGATIONTYPE_OBJECTS = 2 }
	Flag based types of ray trace interogation. More...
enum	TIMEofday { TIMEofday_INVALID = -1, TIMEofday_DAWN , TIMEofday_DAY , TIMEofday_DUSK ,

	TIMEOFDAY_NIGHT, NUM_PERIODS_OF_DAY
	}
	General time of day. More...
enum	TIMEREF { TIMEREF_INVALID = -1, TIMEREF_LOCAL, TIMEREF_ZULU, NUM_TIMEREF }
	Used in time queries to reference Zulu (UTC) or Local time. More...
enum	UNITMODE { UNITMODE_SLEEP, UNITMODE_ZOMBIE, UNITMODE_WAYPOINT, UNITMODE_TAKEOFF, UNITMODE_LANDING, UNITMODE_TAXI, UNITMODE_WORKING, UNITMODE_WAITING }
	AI Unit modes. More...
enum	WEATHER_MODE { WEATHER_MODE_THEME, WEATHER_MODE_CUSTOM, WEATHER_MODE_GLOBAL }
enum	THERMAL_VISUAL_TYPE { THERMAL_VISUAL_NONE, THERMAL_VISUAL_NATURAL, THERMAL_VISUAL_SCHEMATIC, THERMAL_VISUAL_MAX = THERMAL_VISUAL_SCHEMATIC }
enum	CLOUD_COVERAGE_DENSITY { CLOUD_COVERAGE_DENSITY_MIN = 5, CLOUD_COVERAGE_DENSITY_LOW = CLOUD_COVERAGE_DENSITY_MIN, CLOUD_COVERAGE_DENSITY_MEDIUM, CLOUD_COVERAGE_DENSITY_HIGH, CLOUD_COVERAGE_DENSITY_MAXIMUM, CLOUD_COVERAGE_DENSITY_MAX = CLOUD_COVERAGE_DENSITY_MAXIMUM }
enum	CLOUD_DRAW_DISTANCE { CLOUD_DRAW_DISTANCE_MIN = 3, CLOUD_DRAW_DISTANCE_60_MILES = CLOUD_DRAW_DISTANCE_MIN, CLOUD_DRAW_DISTANCE_70_MILES, CLOUD_DRAW_DISTANCE_80_MILES, CLOUD_DRAW_DISTANCE_90_MILES, CLOUD_DRAW_DISTANCE_10_MILES, CLOUD_DRAW_DISTANCE_110_MILES, CLOUD_DRAW_DISTANCE_MAX = CLOUD_DRAW_DISTANCE_110_MILES }
enum	CLOUD_TYPE { CLOUD_NONE, CLOUD_CIRRUS, CLOUD_STRATUS, CLOUD_CUMULUS, CLOUD_CUMULONIMBUS, CLOUD_VISIBILITY_LAYER, CLOUD_THUNDERSTORM, CLOUD_MAX }
enum	CLOUD_COVER { CLOUD_CLEAR, CLOUD_FEW_1_8, CLOUD_FEW_2_8, CLOUD_SCATTERED_3_8, CLOUD_SCATTERED_4_8, CLOUD_BROKEN_5_8, CLOUD_BROKEN_6_8, CLOUD_BROKEN_7_8, CLOUD_OVERCAST_8_8, CLOUD_C_MAX }
enum	CLOUD_TOP { CLOUDTOP_FLAT, CLOUDTOP_ROUND, CLOUDTOP_ANVIL, CLOUDTOP_TOP_MAX }
enum	TURBULANCE { TURB_NONE, TURB_LIGHT, TURB_MODERATE, TURB_HEAVY, TURB_SEVERE, TURB_MAX }
enum	PRECIPTYPE { PRECIP_NONE, PRECIP_RAIN, PRECIP_SNOW, PRECIP_MIST }

```

enum PRECIPRATE { PRECIPRATE_NONE, PRECIPRATE_LOW, PRECIPRATE_MODERATE,
PRECIPRATE_HIGH, PRECIPRATE_VHIGH, PRECIPRATE_MAX
}

enum ICINGRATE { ICINGRATE_NONE, ICINGRATE_TRACE, ICINGRATE_LIGHT,
ICINGRATE_MODERATE, ICINGRATE_SEVERE, ICINGRATE_MAX
}

enum WINDSHEAR { WINDSHEAR_GRADUAL, WINDSHEAR_MODERATE, WINDSHEAR_STEEP,
WINDSHEAR_INSTANTANEOUS, WINDSHEAR_MAX
}

```

Macro Definition Documentation

◆ DECLARE_IUNKNOWN_WITH_INLINE_REFCOUNT_IMPL

```
#define DECLARE_IUNKNOWN_WITH_INLINE_REFCOUNT_IMPL()
```

Value:

```
public: \
    DEFAULT_REFCOUNT_INLINE_IMPL() \
    virtual HRESULT STDMETHODCALLTYPE QueryInterface(_in REFIID riid,
        out void **ppvObject); \
```

◆ DEFAULT_REFCOUNT_INLINE_IMPL

```
#define DEFAULT_REFCOUNT_INLINE_IMPL()
```

Value:

```
public: \
    volatile LONG m_RefCount; \
    virtual ULONG STDMETHODCALLTYPE AddRef() {return
        InterlockedIncrement(&m_RefCount); } \
    virtual ULONG STDMETHODCALLTYPE Release() \
    { \
        ULONG RetVal = InterlockedDecrement(&m_RefCount); \
        if (RetVal == 0) {delete this;} \
        else if (RetVal & 0x80000000) {__debugbreak();} \
        return RetVal; \
    } \
```

◆ DXYZ

```
#define DXYZ P3DDXYZ
```

Simple vector structures for passing between **P3D** host and implementation.

◆ FXYZ

```
#define FXYZ P3DFXYZ
```

◆ NO_COPY_IUNKNOWN_IMPL

```
#define NO_COPY_IUNKNOWN_IMPL( IClassName )
```

Value:

```
private: \
    IClassName( const IClassName& ); \
    IClassName& operator=( const IClassName& );
```

Typedef Documentation

◆ INameList

```
using INameList = IListBuilder<const WCHAR>
```

◆ PEventCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PEventCallback) (_in ISimObject &Sim, __in
double dProperty, __in int iIndex)
```

Event input callback.

◆ PEventCallbackWithSubString

```
typedef HRESULT(STDMETHODCALLTYPE * PEventCallbackWithSubString) (_in ISimObject
&Sim, __in double dProperty, __in LPCWSTR pszSecondarySecondarySubstring, __in int iIndex)
```

Event input callback for double (with secondary string input)

◆ PEventStringCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PEventStringCallback) (_in ISimObject &Sim,
__in LPCWSTR pszProperty, __in int iIndex)
```

Event input callback.

◆ PEventVectorCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PEventVectorCallback) (_in ISimObject &Sim,
__in const DXYZ &vProperty, __in int iIndex)
```

Event input callback.

◆ PNewScenarioNotify

```
typedef void(STDMETHODCALLTYPE * PNewScenarioNotify) (BOOL bOnSave)
```

Function callback for notification that a new scenario is loaded or saved.

These are registered through the IScenarioManager interface

◆ POnObjectCreateCallback

```
typedef HRESULT(STDMETHODCALLTYPE * POnObjectCreateCallback) (__in IUnknown &Obj)
```

Register callback on object creation. This could be a simobject or a library object.

◆ POnObjectRemoveCallback

```
typedef HRESULT(STDMETHODCALLTYPE * POnObjectRemoveCallback) (__in IUnknown &Obj)
```

Register callback on object removal. This could be a simobject or a library object.

◆ POnUserObjectChangedCallback

```
typedef HRESULT(STDMETHODCALLTYPE * POnUserObjectChangedCallback) (__in IUnknown &NewObj, __in IUnknown &OldObj)
```

Register callback on user object changing from one user to another (during runtime, not scenario load or vehicle change via UI)

◆ PPropertyCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PPropertyCallback) (__in const ISimObject &Sim, __out double &dProperty, __in int iIndex)
```

Double property callback.

◆ PPropertyCallbackWithSubString

```
typedef HRESULT(STDMETHODCALLTYPE * PPropertyCallbackWithSubString) (__in const ISimObject &Sim, __out double &dProperty, __in LPCWSTR pszSecondarySubstring, __in int iIndex)
```

Double property callback (with secondary string input)

◆ PPropertyStringCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PPropertyStringCallback) (_in const ISimObject &Sim, _out LPWSTR pszProperty, _in UINT uStringLength, _in int iIndex)
```

String property callback.

◆ PPropertyVectorCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PPropertyVectorCallback) (_in const ISimObject &Sim, _out DXYZ &vProperty, _in int iIndex)
```

Vector (double x,y,z) property callback.

◆ PSaveLoadCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PSavedLoadCallback) (_in LPCWSTR pszSection, _in unsigned int ulnstance, _in LPCWSTR pszKeyword, _inout void *pvVal, _in const SAVED_DATA_TYPE eDataType)
```

Function pointer for state save/load (.FXML files) This pointer is passed to each ISimulation when it's time for save/load Section names will automatically be constructed as:

[SectionName.Instance.SimObjectID] allowing for multiple instances of a system (e.g. multi-engines) Constructed section names are limited to a maximum of 128 characters.

SAVED_DATA_TYPE allows you to distinguish your data between numeric and string values

◆ PSimCreateFunc

```
typedef HRESULT(STDMETHODCALLTYPE * PSimCreateFunc) (_in __notnull IBaseObjectV400 *, _out __notnull ISimObject **)
```

Function pointer prototype for object implementation creation function. This function is called for each instance of the respective simobject class

Enumeration Type Documentation

◆ CLOUD_COVER

enum CLOUD_COVER

Enumerator

CLOUD_CLEAR

CLOUD_FEW_1_8
CLOUD_FEW_2_8
CLOUD_SCATTERED_3_8
CLOUD_SCATTERED_4_8
CLOUD_BROKEN_5_8
CLOUD_BROKEN_6_8
CLOUD_BROKEN_7_8
CLOUD_OVERCAST_8_8
CLOUD_C_MAX

◆ CLOUD_COVERAGE_DENSITY

enum **CLOUD_COVERAGE_DENSITY**

Enumerator

CLOUD_COVERAGE_DENSITY_MIN
CLOUD_COVERAGE_DENSITY_LOW
CLOUD_COVERAGE_DENSITY_MEDIUM
CLOUD_COVERAGE_DENSITY_HIGH
CLOUD_COVERAGE_DENSITY_MAXIMUM
CLOUD_COVERAGE_DENSITY_MAX

◆ CLOUD_DRAW_DISTANCE

enum **CLOUD_DRAW_DISTANCE**

Enumerator

CLOUD_DRAW_DISTANCE_MIN
CLOUD_DRAW_DISTANCE_60_MILES
CLOUD_DRAW_DISTANCE_70_MILES
CLOUD_DRAW_DISTANCE_80_MILES
CLOUD_DRAW_DISTANCE_90_MILES
CLOUD_DRAW_DISTANCE_10_MILES
CLOUD_DRAW_DISTANCE_110_MILES
CLOUD_DRAW_DISTANCE_MAX

◆ CLOUD_TOP

enum **CLOUD_TOP**

Enumerator

CLOUDTOP_FLAT
CLOUDTOP_ROUND
CLOUDTOP_ANVIL
CLOUDTOP_TOP_MAX

◆ CLOUD_TYPE

enum CLOUD_TYPE

Enumerator

CLOUD_NONE	
CLOUD_CIRRUS	
CLOUD_STRATUS	
CLOUD_CUMULUS	
CLOUD_CUMULONIMBUS	
CLOUD_VISIBILITY_LAYER	
CLOUD_THUNDERSTORM	
CLOUD_MAX	

◆ COLLISIONTYPE

enum COLLISIONTYPE

COLLISIONTYPE - Object-to-object collision results

- Returned in IBaseObject::CheckCollision

Enumerator

COLLISIONTYPE_NONE	
COLLISIONTYPE_SIMOBJECT	
COLLISIONTYPE_BUILDING	
COLLISIONTYPE_TERRAIN	
COLLISIONTYPE_OTHER	

◆ DYNAMIC_LIGHT

enum DYNAMIC_LIGHT

Dynamic light type enum.

Enumerator

DYNAMIC_LIGHT_POINT	
DYNAMIC_LIGHT_SPOT	

◆ EVENTTYPE

enum EVENTTYPE

Type of keyboard or joystick mapping associated with an event being registered.

Enumerator

EVENTTYPE_NORMAL
EVENTTYPE_AXIS
EVENTTYPE_POV
EVENTTYPE_NORMAL_NATIVE_OVERRIDE
EVENTTYPE_AXIS_NATIVE_OVERRIDE
EVENTTYPE_POV_NATIVE_OVERRIDE
EVENTTYPE_NOT_MAPPABLE

◆ HORIZONTAL_ALIGNMENT

enum **HORIZONTAL_ALIGNMENT**

Text horizontal alignment enum.

Enumerator

HORIZONTAL_ALIGNMENT_LEFT
HORIZONTAL_ALIGNMENT_CENTER
HORIZONTAL_ALIGNMENT_RIGHT

◆ ICINGRATE

enum **ICINGRATE**

Enumerator

ICINGRATE_NONE
ICINGRATE_TRACE
ICINGRATE_LIGHT
ICINGRATE_MODERATE
ICINGRATE_SEVERE
ICINGRATE_MAX

◆ INTERROGATIONTYPE

enum **INTERROGATIONTYPE**

Flag based types of ray trace interogation.

Enumerator

INTERROGATIONTYPE_NONE
INTERROGATIONTYPE_TERRAIN
INTERROGATIONTYPE_OBJECTS

◆ NET_MODE_TYPE

enum NET_MODE_TYPE

Types of network modes for a given object.

Enumerator

NET_MODE_TYPE_NORMAL
NET_MODE_TYPE_REMOTE
NET_MODE_TYPE_MASTER
NET_MODE_TYPE_SLAVE

◆ PRECIPRATE

enum PRECIPRATE

Enumerator

PRECIPRATE_VLOW
PRECIPRATE_LOW
PRECIPRATE_MODERATE
PRECIPRATE_HIGH
PRECIPRATE_VHIGH
PRECIPRATE_MAX

◆ PRECIPTYPE

enum PRECIPTYPE

Enumerator

PRECIP_NONE
PRECIP_RAIN
PRECIP_SNOW
PRECIP_MAX

◆ PROPERTY_TYPE

enum PROPERTY_TYPE

Types of properties.

Enumerator

PROPERTY_TYPE_EVENT
PROPERTY_TYPE_EVENT_WITH_SUBSTRING_INPUT
PROPERTY_TYPE_EVENT_VECTOR
PROPERTY_TYPE_EVENT_STRING
PROPERTY_TYPE_DOUBLE
PROPERTY_TYPE_DOUBLE_WITH_SUBSTRING_INPUT

PROPERTY_TYPE_STRING
PROPERTY_TYPE_VECTOR
NUM_PROPERTY_TYPES

◆ SAVED_DATA_TYPE

enum SAVED_DATA_TYPE

Types of data saved via the PSaveLoadCallback.

Enumerator

SAVED_DATA_TYPE_DOUBLE
SAVED_DATA_TYPE_STRING
NUM_SAVED_DATA_TYPES

◆ TEXT_FONT

enum TEXT_FONT

Text font type enum.

Enumerator

TEXT_FONT_DEFAULT
TEXT_FONT_SMALL
TEXT_FONT_SMALL_BOLD
TEXT_FONT_MEDIUM_BOLD
TEXT_FONT_LARGE_BOLD
TEXT_FONT_XLARGE_BOLD

◆ THERMAL_VISUAL_TYPE

enum THERMAL_VISUAL_TYPE

Enumerator

THERMAL_VISUAL_NONE
THERMAL_VISUAL_NATURAL
THERMAL_VISUAL_SCHEMATIC
THERMAL_VISUAL_MAX

◆ TIMEOFDAY

enum TIMEOFDAY

General time of day.

Enumerator

TIMEOFDAY_INVALID	
TIMEOFDAY_DAWN	
TIMEOFDAY_DAY	
TIMEOFDAY_DUSK	
TIMEOFDAY_NIGHT	
NUM_PERIODS_OF_DAY	

◆ TIMEREF

enum **TIMEREF**

Used in time queries to reference Zulu (UTC) or Local time.

Enumerator

TIMEREF_INVALID	
TIMEREF_LOCAL	
TIMEREF_ZULU	
NUM_TIMEREF	

◆ TURBULANCE

enum **TURBULANCE**

Enumerator

TURB_NONE	
TURB_LIGHT	
TURB_MODERATE	
TURB_HEAVY	
TURB_SEVERE	
TURB_MAX	

◆ UNITMODE

enum **UNITMODE**

AI Unit modes.

Enumerator

UNITMODE_SLEEP	
UNITMODE_ZOMBIE	
UNITMODE_WAYPOINT	
UNITMODE_TAKEOFF	
UNITMODE_LANDING	
UNITMODE_TAXI	

UNITMODE_WORKING

UNITMODE_WAITING

◆ VERTICAL_ALIGNMENT

enum VERTICAL_ALIGNMENT

Text vertical alignment enum.

Enumerator

VERTICAL_ALIGNMENT_TOP

VERTICAL_ALIGNMENT_CENTER

VERTICAL_ALIGNMENT_BOTTOM

◆ WEATHER_MODE

enum WEATHER_MODE

Enumerator

WEATHER_MODE_THEME

WEATHER_MODE_CUSTOM

WEATHER_MODE_GLOBAL

◆ WINDSHEAR

enum WINDSHEAR

Enumerator

WINDSHEAR_GRADUAL

WINDSHEAR_MODERATE

WINDSHEAR_STEEP

WINDSHEAR_INSTANTANEOUS

WINDSHEAR_MAX

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

ISimObject

Overview

The SimObject API utilizes a service-based methodology for building simulation behaviors to be visualized in Prepar3D. The API enables a solution developer to create a simulation object (SimObject) complete with customized behaviors, input properties (also referred to as events or triggers), and state properties (also referred to as simvars or simply properties). These properties can be referenced in content such as SimObject gauges, animations, and scenario scripts which are discussed in more detail in other parts of the SDK. The properties are text-based, and can be referenced in the same way as the stock simvars and events are referred to in other parts of the Prepar3D SDK.

Types.h

Includes common data types found throughout the ISimObject Samples.

PSaveLoadCallback() - Function pointer for state save/load (.FXML files). This pointer is passed to each ISimulation when it's time for save/load Section names will automatically be constructed as: [SectionName.Instance.SimObjectID] allowing for multiple instances of a system (e.g. multi-engines) Constructed section names are limited to a maximum of 128 characters. SIM_DATA_TYPE allows saving either numeric or string data.

```
typedef HRESULT (STDMETHODCALLTYPE *PSaveLoadCallback)(__in LPCTSTR  
    pszSection, __in unsigned int uInstance, __in LPCTSTR pszKeyword,  
    __out void* pvVal, __in const SAVED DATA TYPE eDataType);\\n\\n
```

The property type enum used in the property string->ID lookup. See IBaseObject.

```
typedef enum  
{  
    PROPERTY_TYPE_EVENT,  
    PROPERTY_TYPE_EVENT_WITH_SUBSTRING_INPUT,  
    PROPERTY_TYPE_EVENT_VECTOR,  
    PROPERTY_TYPE_EVENT_STRING,  
    PROPERTY_TYPE_DOUBLE,  
    PROPERTY_TYPE_DOUBLE_WITH_SUBSTRING_INPUT,  
    PROPERTY_TYPE_STRING,  
    PROPERTY_TYPE_VECTOR,  
    NUM_PROPERTY_TYPES,  
    PROPERTY_TYPE;
```

Enum definition for data types used when saving / loading. See ISimulation and PSaveLoadCallback.

```
typedef enum  
{  
    SAVED_DATA_TYPE_DOUBLE,  
    SAVED_DATA_TYPE_STRING,  
    NUM_SAVED_DATA_TYPES  
} SAVED DATA TYPE;
```

Enum definition for the various modes in which an Artificially Intelligent (AI) object can be.

```
typedef enum  
{  
    UNITMODE_SLEEP,  
    UNITMODE_ZOMBIE,  
    UNITMODE_WAYPOINT,  
    UNITMODE_TAKEOFF,  
    UNITMODE_LANDING,  
    UNITMODE_TAXI,  
    UNITMODE_WORKING,  
    UNITMODE_WAITING,  
} UNITMODE_TYPE;
```

A **BasicWaypoint** is generally used to define a point along a path. It is used primarily by the AI system.

```
class BasicWaypoint  
{  
public:  
    DXYZ vLonAltLat; //Lat/Lon (Radians), Alt (Feet)  
    double dHeading; //Radians  
};
```

Enum definition for various network modes. See Object Mode Monitoring.

```
typedef enum  
{  
    NET_MODE_TYPE_NORMAL, // The object is owned by the current client  
    NET_MODE_TYPE_REMOTE, // The object is owned by another client  
    NET_MODE_TYPE_MASTER, // Shared Cockpit: The object is owned by the
```

```
    current_client  
    NET_MODE_TYPE_SLAVE,      // Shared Cockpit: The object is owned by  
    another client  
} NET_MODE_TYPE;
```

Versioning Code

Compiled with the PDK and ISimObject interfaces will be expected to function in subsequent versions of the SDK. To do so, each interface name is appended with the version number, and is derived from the preceding version. The preceding versions will be maintained intact in the Legacy subfolder. It is recommended that all new code utilizes the latest version when defining your objects and each QueryInterface supports all versions. Internal to Prepar3D, the earliest version possible will be used.

Namespaces

P3D

Prepar3D SDK namespace used primarily for the PDK and its services.

Classes

```
class ISimObjectManagerV500  
class ISimObjectV440  
class ISimulationV310  
class IBaseObjectV450  
class ISubSystemFactoryV500  
class WorldConstants  
class SurfaceInfoV400  
class WeatherInfoV400  
class IMassPropertiesV01  
class IForceMomentsV01  
class ICollisionServiceV01  
class IAircraftServiceV01  
class IAirplaneServiceV01  
class IRotorcraftServiceV01  
class IBoatServiceV01  
class IGroundVehicleServiceV01  
class IAtcServiceV01  
class IRadarSignatureServiceV01  
class IDoorServiceV01  
class IFuelServiceV400  
class ISurfaceQueryManagerV400  
class IWaypointQueryManagerV400  
class IAvatarSimV01  
class IAnimationControllerV01  
class IAvatarAttachServiceV01  
class IMarkerManagerV310  
class IDesignatorServiceV340  
class IRayTraceManagerV340  
class IEmissionsServiceV340  
class IRadioSystemV400  
class IAttachmentServiceV430  
class IAIBehaviorManagerV01  
class IAIBehaviorWingmanFormationV01  
class IAIBehaviorAttackerV400  
class IAIBehaviorPursueV500  
class IAIBehaviorCombatAirPatrolV01  
class IAIBehaviorCloseAirSupportV01  
class IAIBehaviorSearchTrackV01  
class ISimObjectAIV02  
class IAIRobotAIServiceV02  
class IHelicopterAIServiceV420  
class IGroundVehicleAIServiceV01  
class IWeaponsSystemV440  
class IWeaponServiceV420  
class ICountermeasureSystemV01  
class ICountermeasureServiceV02  
class IGunSystemV440  
class IGunV400  
class IFireControlSystemV01  
class IGuidanceSystemV01
```

Class	Description
class IPylonServiceV01	
class ArticulatedPart	
class ArticulatedParameter	
class IPduBuilderV440	
class IPduReaderV440	
class IPduCallbackV440	
class IDISManagerV450	
class IDISServiceV400	
union ArticulatedParameter._unnamed_	

Class Documentation

P3D::ISimObjectManagerV500

class P3D::ISimObjectManagerV500

Handles tasks that are not associated with an instance of a simobject. This includes:

- Registration of implementation factories and associated properties
- Global application properties (e.g. world constants)
- Object queries

Inherits ISimObjectManagerV440.

Private Member Functions

```

virtual HRESULT RegisterSimulationCategory (_in GUID guidCategory, _in LPCWSTR pszCategoryName, _in
                                         _notnull PSimCreateFunc pcbCreateFunction) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in LPCWSTR
                                 pszPropertyBaseUnits, _in _notnull PPropertyCallback pcbProperty) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in LPCWSTR
                                 pszPropertyBaseUnits, _in _notnull PPropertyVectorCallback pcbProperty) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in _notnull
                                 PPropertyStringCallback pcbProperty) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in LPCWSTR
                                 pszPropertyBaseUnits, _in _notnull PPropertyCallbackWithSubString pcbProperty) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in LPCWSTR
                                 pszPropertyBaseUnits, _in _notnull PEventCallback pcbEvent, _in EVENTTYPE eType) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in LPCWSTR
                                 pszPropertyBaseUnits, _in _notnull PEventVectorCallback pcbEvent) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in _notnull
                                 PEventStringCallback) PURE
virtual HRESULT RegisterProperty (_in GUID guidCategory, _in LPCWSTR pszPropertyName, _in _notnull
                                 PEventCallbackWithSubString pcbEvent) PURE
virtual HRESULT GetWorldConstants (_out WorldConstants &) const PURE
virtual HRESULT GetUnitCode (_in LPCWSTR pszPropertyUnits, _out int &iUnitCode) const PURE
virtual HRESULT GetObject (_in UINT idObject, _out IBaseObjectV400 **ppvObject) const PURE
virtual HRESULT GetObject (_in UINT idObject, _in REFIID iid, _out void **ppvObject) const PURE
virtual HRESULT GetUserObject (_out IBaseObjectV400 **ppvUserObject) const PURE
virtual HRESULT GetUserObject (_in REFIID iid, _out void **ppvUserObject) const PURE
virtual HRESULT RegisterOnObjectCreateCallback (_in _notnull POnObjectCreateCallback pCb) PURE
virtual HRESULT UnRegisterOnObjectCreateCallback (_in _notnull POnObjectCreateCallback pCb) PURE
virtual HRESULT RegisterOnObjectRemoveCallback (_in _notnull POnObjectRemoveCallback pCb) PURE
virtual HRESULT UnRegisterOnObjectRemoveCallback (_in _notnull POnObjectRemoveCallback pCb) PURE
virtual HRESULT RegisterOnUserObjectChangedCallback (_in _notnull POnUserObjectChangedCallback pCb)
                                         PURE
virtual HRESULT UnRegisterOnUserObjectChangedCallback (_in _notnull POnUserObjectChangedCallback
                                         pCb) PURE
virtual HRESULT GetObjectsInRadius (_in const DXYZ &vLonAltLat, _in float fRadiusFeet, _inout UINT
                                         &nObjects, _out UINT *rgObjectIDs) const PURE
virtual HRESULT GetNonTrafficObjectsInRadius (_in const DXYZ &vLonAltLat, _in float fRadiusFeet, _inout
                                         UINT &nObjects, _out UINT *rgObjectIDs) const PURE
virtual float GetRealismSetting () const PURE
virtual BOOL IsCrashDetectionOn () const PURE
virtual BOOL IsCollisionBetweenObjectsOn () const PURE
virtual float GetCrashToleranceScalar () const PURE
virtual HRESULT RemoveObject (_in UINT idObject) PURE
virtual HRESULT CreateObject (_in _notnull LPCWSTR pszTitle, _out UINT &idObject) PURE
virtual HRESULT GetUserAvatar (_out IBaseObjectV400 **ppvUserAvatar) const PURE
virtual HRESULT GetUserAvatar (_in REFIID iid, _out void **ppvUserAvatar) const PURE
virtual uint GetNumberOfCategories () const PURE
virtual HRESULT GetCategoryId (_out GUID &guidCategoryId, _out _notnull LPWSTR pszCategoryFriendlyName,
                             _in uint uNameLen, _out BOOL &blsNativeSimulation, _in uint iIndex) const PURE

```

Member Function Documentation

◆ CreateObject()

```
virtual HRESULT CreateObject( __in __notnull LPCWSTR pszTitle,  
                           __out UINT & idObject  
                           )
```

private virtual

Attempts to create an object with the given container title.

Parameters

pszTitle The container title object to be created.
idObject The object id of the newly created object.

Returns

S_OK if the object was successfully created, E_FAIL otherwise.

◆ GetCategoryId()

```
virtual HRESULT GetCategoryId( __out GUID & guidCategoryId,  
                           __out __notnull LPWSTR pszCategoryFriendlyName,  
                           __in UINT uNameLen,  
                           __out BOOL & blsNativeSimulation,  
                           __in UINT iIndex  
                           )
```

private virtual

Gets the simulation category unique GUID and friendly string name.

Parameters

iIndex The unique (0-based) index into the list of registered categories.
guidCategoryId The GUID id unique to this simulation category.
pszCategoryFriendlyName The friendly string name for the category. These are guaranteed to be unique only for native simulations in core Prepar3D.
uNameLen The maximum length of the allocated string friendly name.
blsNativeSimulation Indicates if this simulation is natively implemented in core Prepar3D (TRUE) or externally developed (FALSE).

Returns

S_OK if the category is successfully found. E_INVALIDARG if the index exceeds the maximum index of the registered simulation list.

Remarks

Indexes are guaranteed to remain unique for the lifetime of a Prepar3D instance.

◆ GetCrashToleranceScalar()

```
virtual float GetCrashToleranceScalar( ) const
```

private virtual

The user-selected scalar for determining crash tolerance

◆ GetNonTrafficObjectsInRadius()

```
virtual HRESULT GetNonTrafficObjectsInRadius( __in const DXYZ & vLonAltLat,  
                                           __in float fRadiusFeet,  
                                           __inout UINT & nObjects,  
                                           __out UINT * rgObjectIDs  
                                           )
```

const private virtual

Returns a list of object IDs for a given radius. Does not include traffic

Parameters

nObjects IN: The max number of elements requested. This must be no smaller than the size of the array pointed to by rgObjectIDs
nObjects OUT: The actual number of objects found.
rgObjectIDs Address of array in which object IDs are returned.
NOTE: It is the callers responsibility to allocate the arrays required memory.

◆ GetNumberOfCategories()

```
virtual UINT GetNumberOfCategories( ) const
```

private virtual

Gets the number of registered simulations.

Returns

Number of registered simulation categories

Remarks

Note that if this queried at startup during DLL loading, some externally developed categories may not yet be registered. Core Prepar3D native simulations will be registered by that time.

◆ GetObject() [1/2]

```
virtual HRESULT GetObject ( __in UINT idObject,
                           __out IBaseObjectV400 ** ppObject
                           ) const [private] [virtual]
```

Gets another IBaseObject ref for a given ID

◆ GetObject() [2/2]

```
virtual HRESULT GetObject ( __in UINT idObject,
                           __in REFIID riid,
                           __out void ** ppvObject
                           ) const [private] [virtual]
```

Gets another specific version of an IBaseObject ref for a given ID

◆ GetObjectsInRadius()

```
virtual HRESULT GetObjectsInRadius ( __in const DXYZ & vLonAltLat,
                                    __in float fRadiusFeet,
                                    __inout UINT & nObjects,
                                    __out UINT * rgObjectIDs
                                    ) const [private] [virtual]
```

Returns a list of object IDs for a given radius.

Parameters

nObjects IN: the max number of elements requested. This must be no smaller than the size of the array pointed to by **rgObjectIDs**.
nObjects OUT: the actual number of objects found.
rgObjectIDs address of array in which object IDs are returned.
NOTE: It is the callers responsibility to allocate the array's required memory

◆ GetRealismSetting()

```
virtual float GetRealismSetting ( ) const [private] [virtual]
```

The user-selected general realism scalar, where 0.0 is "easy" and 1.0 is "real". This can be used to scale your implementation as appropriate

◆ GetUnitCode()

```
virtual HRESULT GetUnitCode ( __in LPCWSTR pszPropertyUnits,
                            __out int & iUnitCode
                            ) const [private] [virtual]
```

Decodes a string units to its integer ID. This can be useful to get at initialization as it is less performant to query properties using the string version. e.g. "feet per second" to ID.

◆ GetUserAvatar() [1/2]

```
virtual HRESULT GetUserAvatar ( __out IBaseObjectV400 ** ppUserAvatar ) const [private] [virtual]
```

Gets an IBaseObject ref for the current user avatar.

Parameters

ppUserAvatar The IBaseObject ref for the current user avatar.

Returns

S_OK if the object was successfully found, E_FAIL otherwise.

Remarks

The user avatar and the user object may be the same if the user has selected an avatar object.

◆ GetUserAvatar() [2/2]

```
virtual HRESULT GetUserAvatar ( __in REFIID riid,
                               __out void ** ppvUserAvatar
                               ) const [private] [virtual]
```

Gets an IBaseObject ref for the current user avatar.

Parameters

ppvUserAvatar The IBaseObject ref for the current user avatar.
riid Interface ID.

Returns

S_OK if the object was successfully found, E_FAIL otherwise.

Remarks

The user avatar and the user object may be the same if the user has selected an avatar object.

◆ GetUserObject() [1/2]

```
virtual HRESULT GetUserObject ( __out IBaseObjectV400 ** ppUserObject ) const [private] [virtual]
```

Gets an IBaseObject ref for the current user object.

NOTE: If the user object is the Viewer and there is a previous user object, this will return the previous user object. Otherwise, it will return the Viewer.

◆ GetUserObject() [2/2]

```
virtual HRESULT GetUserObject ( __in REFIID riid,  
                               __out void ** ppvUserObject  
                           ) const [private] [virtual]
```

Gets a specific version IBaseObject ref for the current user object.

NOTE: If the user object is the Viewer and there is a previous user object, this will return the previous user object. Otherwise, it will return the Viewer.

◆ GetWorldConstants()

```
virtual HRESULT GetWorldConstants ( __out WorldConstants & ) const [private] [virtual]
```

World Constants are constant values describing the Earth atmosphere and geometry.

NOTE: While this is accessed through IBaseObject, these values will be constant for all SimObjects, and a single static copy could be shared across multiple instances.

◆ IsCollisionBetweenObjectsOn()

```
virtual BOOL IsCollisionBetweenObjectsOn ( ) const [private] [virtual]
```

The user-selected flag for whether to detect crashes between simobjects or not

◆ IsCrashDetectionOn()

```
virtual BOOL IsCrashDetectionOn ( ) const [private] [virtual]
```

The user-selected flag that dictates whether to process a crash or not

◆ RegisterOnObjectCreateCallback()

```
virtual HRESULT RegisterOnObjectCreateCallback ( __in __notnull POnObjectCreateCallback pCb ) [private] [virtual]
```

Used to register a callback function that is called upon creation of any new object. See [types.h](#) for callback definition.

◆ RegisterOnObjectRemoveCallback()

```
virtual HRESULT RegisterOnObjectRemoveCallback ( __in __notnull POnObjectRemoveCallback pCb ) [private] [virtual]
```

Used to register a callback function that is called upon destruction of any existing object. The call is just prior to destruction. See [types.h](#) for callback definition.

◆ RegisterOnUserObjectChangedCallback()

```
virtual HRESULT RegisterOnUserObjectChangedCallback ( __in __notnull POnUserObjectChangedCallback pCb ) [private] [virtual]
```

Used to register a callback function that is called whenever the user is moved from one object to another. See [types.h](#) for callback definition.

◆ RegisterProperty() [1/8]

```
virtual HRESULT RegisterProperty ( __in GUID guidCategory,  
                                 __in LPCWSTR pszPropertyName,  
                                 __in LPCWSTR pszPropertyBaseUnits,  
                                 __in __notnull PPropertyCallback pcbProperty )
```

)

private virtual

Property "simvar" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: Double

◆ RegisterProperty() [2/8]

```
virtual HRESULT RegisterProperty(_in GUID guidCategory,
                                _in LPCWSTR pszPropertyName,
                                _in LPCWSTR pszPropertyBaseUnits,
                                _in __notnull PPropertyVectorCallback pcbProperty
                               )
```

private virtual

Property "simvar" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: Vector (DXYZ)

◆ RegisterProperty() [3/8]

```
virtual HRESULT RegisterProperty(_in GUID guidCategory,
                                _in LPCWSTR pszPropertyName,
                                _in __notnull PPropertyStringCallback pcbProperty
                               )
```

private virtual

Property "simvar" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: String

◆ RegisterProperty() [4/8]

```
virtual HRESULT RegisterProperty(_in GUID guidCategory,
                                _in LPCWSTR pszPropertyName,
                                _in LPCWSTR pszPropertyBaseUnits,
                                _in __notnull PPropertyCallbackWithSubString pcbProperty
                               )
```

private virtual

Property "simvar" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: Double (with secondary substring input)

◆ RegisterProperty() [5/8]

```
virtual HRESULT RegisterProperty(_in GUID guidCategory,
                                _in LPCWSTR pszPropertyName,
                                _in LPCWSTR pszPropertyBaseUnits,
                                _in __notnull PEventCallback pcbEvent,
                                _in EVENTTYPE eType
                               )
```

private virtual

Property "event" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: Event

◆ RegisterProperty() [6/8]

```
virtual HRESULT RegisterProperty(_in GUID guidCategory,
                                _in LPCWSTR pszPropertyName,
                                _in LPCWSTR pszPropertyBaseUnits,
                                _in __notnull PEventVectorCallback pcbEvent
                               )
```

private virtual

Property "event" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: Event vector

◆ RegisterProperty() [7/8]

```
virtual HRESULT RegisterProperty ( __in GUID guidCategory,
                                 __in LPCWSTR pszPropertyName,
                                 __in __notnull PEventStringCallback
                               )
```

private virtual

Property "event" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: Event string

◆ RegisterProperty() [8/8]

```
virtual HRESULT RegisterProperty ( __in GUID guidCategory,
                                 __in LPCWSTR pszPropertyName,
                                 __in LPCWSTR pszPropertyBaseUnits,
                                 __in __notnull PEventCallbackWithSubString pcbEvent
                               )
```

private virtual

Property "event" registration. For a specific simobject implementation (guid), associates: property string name, units, and callback pointer (defined above)

Note

Input: Event double (with secondary substring input)

◆ RegisterSimulationCategory()

```
virtual HRESULT RegisterSimulationCategory ( __in GUID guidCategory,
                                             __in LPCWSTR pszCategoryName,
                                             __in __notnull PSimCreateFunc pcbCreateFunction
                                           )
```

private virtual

Registers an ISimObject implementation at load time with: unique ID, friendly category name (e.g. "airplane"), and factory function pointer. The "pszCategoryName" is a high-level categorization used primarily for UI (e.g. "airplane"). Mainly, it is used as a filter to exclude objects from appearing in the Vehicle Select screen. If you create a unique category name, ensure you add the name to the User Objects key in the Prepar3D.cfg's [Main] section.

◆ RemoveObject()

```
virtual HRESULT RemoveObject ( __in UINT idObject ) private virtual
```

RemoveObject - Remove any local non-user object

Parameters

idObject The id of the object to remove.

◆ UnRegisterOnObjectCreateCallback()

```
virtual HRESULT UnRegisterOnObjectCreateCallback ( __in __notnull POnObjectCreateCallback pCb ) private virtual
```

Used to unregister a callback function previously registered with [RegisterOnObjectCreateCallback\(\)](#)

◆ UnRegisterOnObjectRemoveCallback()

```
virtual HRESULT UnRegisterOnObjectRemoveCallback ( __in __notnull POnObjectRemoveCallback pCb ) private virtual
```

Used to unregister a callback function previously registered with [RegisterOnObjectRemoveCallback\(\)](#)

◆ UnRegisterOnUserObjectChangedCallback()

```
virtual HRESULT UnRegisterOnUserObjectChangedCallback ( __in __notnull POnUserObjectChangedCallback pCb ) private virtual
```

Used to unregister a callback function previously registered with [RegisterOnUserObjectChangedCallback\(\)](#)

◆ P3D::ISimObjectV440

class P3D::ISimObjectV440

Interface from which object implementations must derive.

Inherits ISimObjectV400.

Private Member Functions

```
virtual HRESULT LoadConstantData (__inout void **ppConstantData) PURE
virtual HRESULT UnloadConstantData (__inout void **ppConstantData) PURE
virtual HRESULT LoadDynamicData () PURE
virtual HRESULT Init () PURE
virtual HRESULT DeInit () PURE
    virtual BOOL SupportsLabels () const PURE
virtual HRESULT SetSupportsLabels (BOOL bOn) PURE
    virtual void OnModeChange (int bfNewModes) PURE
    virtual void OnPositionInit () PURE
    virtual void OnSpeedInit (float fSpeed) PURE
virtual HRESULT QueryBaseObject (REFID rid, void **ppv) PURE
virtual HRESULT GetMainSimRate (__out float &fSimRate) const PURE
virtual HRESULT GetMainMinMaxSimRates (__out float &fMinSimRate, __out float
&fMaxSimRate) const PURE
virtual HRESULT SetMainMinMaxSimRates (__in float fMinSimRate, __in float fMaxSimRate)
PURE
```

Member Function Documentation

◆ DeInit()

```
virtual HRESULT DeInit () [private] [virtual]
```

Can be used to release inter-system references prior to the object being destroyed.

◆ GetMainMinMaxSimRates()

```
virtual HRESULT GetMainMinMaxSimRates ( __out float & fMinSimRate,
__out float & fMaxSimRate
) [const] [private] [virtual]
```

Provide the minimum and maximum main simulation rate (Hz). Typically the world position update rate.

◆ GetMainSimRate()

```
virtual HRESULT GetMainSimRate ( __out float & fSimRate ) const [private] [virtual]
```

Provide the main simulation rate (Hz). Typically the world position update rate. This assumes the simulation is registered at a constant simulation rate. Beginning with V440 of this interface, a min and max rate can be used if desired, so the following GetMainSimRate may give more accurate information.

◆ Init()

```
virtual HRESULT Init () [private] [virtual]
```

An appropriate place to initialize data and establish references between subsystems.

◆ LoadConstantData()

```
virtual HRESULT LoadConstantData ( __inout void ** ppConstantData ) [private] [virtual]
```

Where your object class should load data from the disk. The return data is cached for subsequent instances of this same object.

◆ LoadDynamicData()

```
virtual HRESULT LoadDynamicData () [private] [virtual]
```

Called on each object instance. This would be an appropriate place to create the object's runtime subsystems.

◆ OnModeChange()

```
virtual void OnModeChange ( int bfNewModes ) [private] [virtual]
```

Called upon change in modes (pause, slew, etc...)

◆ **OnPositionInit()**

virtual void OnPositionInit() **private** **virtual**

Called whenever Prepar3D has changed the position of this object outside of its own simulation implementation. Examples of this would be positioning from the User Interface, Slew Mode, or terrain resolution changing.

◆ **OnSpeedInit()**

virtual void OnSpeedInit(float fSpeed) **private** **virtual**

Called whenever Prepar3D has changed the speed of this object outside of its own simulation implementation. This would occur normally if positioning from the User Interface. Accessor to get the base object from the ISimObject. NOTE: This previously required a return type of IBaseObjectV400**. Existing implementations will downcast to void** automatically.

◆ **QueryBaseObject()**

virtual HRESULT QueryBaseObject(REFIID riid,
void ** ppv
)
private **virtual**

◆ **SetMainMinMaxSimRates()**

virtual HRESULT SetMainMinMaxSimRates(__in float fMinSimRate,
__in float fMaxSimRate
)
private **virtual**

Sets minimum and maximum main simulation rate (Hz). Typically the world position update rate. This is typically called if there is a desire to synchronize the rates of two or more objects. For example an aircraft and an aircraft carrier, or to prevent perceived jitter when viewing an object from a camera attached to another object. Note: It is the responsibility of the ISimObject developer to determine if the rates are appropriate for the implementation, and subsequently call IBaseObject::RegisterSimulation() (again) with the new rates with the relevant ISimulations.

◆ **SetSupportsLabels()**

virtual HRESULT SetSupportsLabels(**BOOL** bOn) **private** **virtual**

Requests this SimObject to support labels. Return S_OK if the new setting is accepted. If not, return an error code, such as E_FAIL. This value should maintain by this class and returned when requested by SupportsLabel(). You may choose for your class to not support labels. This setting will not override settings in the Traffic Settings.

◆ **SupportsLabels()**

virtual **BOOL** SupportsLabels() const **private** **virtual**

Defines if the SimObject will or will not support labels to be displayed.

◆ **UnloadConstantData()**

virtual HRESULT UnloadConstantData(__inout void ** ppConstantData) **private** **virtual**

Where your object class should unload data from the disk. The return data is cached for subsequent instances of this same object.

◆ **P3D::ISimulationV310**

class P3D::ISimulationV310

Interface to individual simulation subsystems.

Inherits ISimulationV01.

Inherited by **IEngineSystemV500**.

Private Member Functions

virtual HRESULT **Update**(double dDeltaT) **PURE**

virtual HRESULT **SaveLoadState**(__in __notnull **PSaveLoadCallback** pfnCallback, __in const **POD1** hSaveData)

◆ Deserialize()

```
virtual HRESULT Serialize (_in NetOutPublic &netOut) PURE
virtual HRESULT Deserialize (_in NetInPublic &netIn) PURE
```

Member Function Documentation

◆ Deserialize()

```
virtual HRESULT Deserialize (_in NetInPublic & netIn) [private] [virtual]
```

Only called when in an active multiplayer session. This function can be implemented to deserialize network packets that have been sent by other clients for this ISimulation instance. The NetInPublic interface is defined in [NetInOutPublic.h](#).

◆ SaveLoadState()

```
virtual HRESULT SaveLoadState ( _in __notnull PSaveLoadCallback pfnCallback,
                               _in const BOOL bSave
                           ) [private] [virtual]
```

Called when either saving or loading a Prepar3D scenario. The function pointer allows your code to save and load "name - value" pairs in the Prepar3D.xml file. See the definition for PSaveLoadCallback and the supported data type enum SAVED_DATA_TYPE.

◆ Serialize()

```
virtual HRESULT Serialize (_in NetOutPublic & netOut) [private] [virtual]
```

Only called when in an active multiplayer session. This function can be implemented to create network packets that are then broadcast to other clients for this ISimulation instance. The NetOutPublic interface is defined in [NetInOutPublic.h](#)

◆ Update()

```
virtual HRESULT Update ( double dDeltaT ) [private] [virtual]
```

Called by Prepar3D at the iteration rate specified when this ISimulation interface is registered using RegisterSimulation() in the IBaseObject interface.

◆ P3D::IBaseObjectV450

```
class P3D::IBaseObjectV450
```

Object interface on the host side for providing platform information and services for the object
Inherits IBaseObjectV440.

Private Member Functions

```
virtual UINT GetId () const PURE
virtual HRESULT GetMissionId (_out GUID &guid) const PURE
virtual BOOL IsUser () const PURE
virtual UINT GetObjectGroupAssociationId () const PURE
virtual void SetObjectGroupAssociationId (UINT uAssociationId) PURE
virtual BOOL InObjectFoeList (UINT id) const PURE
virtual void SetObjectFoeList (UINT *uEnteredFoeID, UINT size) PURE
virtual BOOL InObjectFriendList (UINT id) const PURE
virtual void SetObjectFriendList (UINT *uEnteredFriendID, UINT size) PURE
virtual int GetMode () const PURE
virtual HRESULT SetCrashMode (double dDeltaT) PURE
virtual HRESULT GetPosition (_out DXYZ &vLonAltLat, _out DXYZ &vPHB, _out DXYZ &vLonAltLatVel, _out DXYZ &vPHBVel) const PURE
virtual HRESULT SetPosition (_in const DXYZ &vLonAltLat, _in const DXYZ &vPHB, _in const DXYZ &vLonAltLatVel, _in const DXYZ &vPHBVel, _in BOOL blsOnGround, _in double dDeltaT) PURE
virtual HRESULT InitPosition (_in const DXYZ *pvLonAltLat, _in const DXYZ *pvPHB, _in const DXYZ *pvLonAltLatVel, _in const DXYZ *pvPHBVel, _in BOOL bSetOnGround) PURE
virtual BOOL IsOnGround () const PURE
virtual HRESULT RotateWorldToBody (_in const DXYZ &vWorld, _out DXYZ &vBody) const PURE
virtual HRESULT RotateBodyToWorld (_in const DXYZ &vBody, _out DXYZ &vWorld) const PURE
virtual HRESULT RegisterSimulation (_in __notnull ISimulation *pSimulation, float fPriority) PURE
```

IUnknown	
virtual HRESULT	RegisterSimulation (_in _notnull ISimulation *pSimulation, float fMinRateHz, float fMaxRateHz) PURE
virtual HRESULT	GetMainMinMaxSimRates (_out float &fMinHz, _out float &fMaxHz) const PURE
virtual HRESULT	SetMainMinMaxSimRates (_in float fMinHz, _in float fMaxHz) PURE
virtual HRESULT	RegisterService (_in REFGUID guidService, _in _notnull IUnknown *punkService) PURE
virtual HRESULT	UnregisterService (_in REFGUID guidService) PURE
virtual HRESULT	GetPropertyCodeAndIndex (_in PROPERTY_TYPE eType, _in LPCWSTR pszPropertyName, _out int &iPropertyCode, _inout int &iIndex) const PURE
virtual HRESULT	GetProperty (_in int iPropertyCode, _in int iUnitCode, _out double &dProperty, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in LPCWSTR pszPropertyName, _in int iUnitCode, _out double &dProperty, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszUnitCode, _out double &dProperty, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in int iPropertyCode, _in int iUnitCode, _out DXYZ &dProperty, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in LPCWSTR pszPropertyName, _in int iUnitCode, _out DXYZ &dProperty, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszUnitCode, _out DXYZ &dProperty, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in int iPropertyCode, _out LPWSTR pszProperty, _in UINT ulLength, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in LPCWSTR pszPropertyName, _out LPWSTR pszProperty, _in UINT ulLength, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in int iPropertyCode, _in LPCWSTR pszSecondarySubstring, _in int iUnitCode, _out double &dProperty, _in int index=0) const PURE
virtual HRESULT	GetProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszSecondarySubstring, _in int iUnitCode, _out double &dProperty, _in int index=0) const PURE
virtual HRESULT	TriggerProperty (_in int iPropertyCode, _in int iUnitCode, _in double dData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in LPCWSTR pszPropertyName, _in int iUnitCode, _in double dData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszUnitCode, _in double dData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in int iPropertyCode, _in int iUnitCode, _in const DXYZ &vData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in LPCWSTR pszPropertyName, _in int iUnitCode, _in const DXYZ &vData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszUnitCode, _in const DXYZ &vData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in int iPropertyCode, _in LPCWSTR pszData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in int iPropertyCode, _in LPCWSTR pszSecondarySubstring, _in int iUnitCode, _in double dData, _in int index) const PURE
virtual HRESULT	TriggerProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszSecondarySubstring, _in int iUnitCode, _in double dData, _in int index) const PURE
virtual HRESULT	RegisterProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszPropertyBaseUnits, _in _notnull PPropertyCallback pcbProperty) PURE
virtual HRESULT	RegisterProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszPropertyBaseUnits, _in _notnull PEventCallback pcbEvent, _in EVENTTYPE eType) PURE
virtual HRESULT	RegisterProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszPropertyBaseUnits, _in _notnull PPropertyVectorCallback pcbProperty) PURE
virtual HRESULT	RegisterProperty (_in LPCWSTR pszPropertyName, _in LPCWSTR pszPropertyBaseUnits, _in _notnull PEventVectorCallback) PURE
virtual HRESULT	RegisterProperty (_in LPCWSTR pszPropertyName, _in _notnull PPropertyStringCallback pcbProperty) PURE
virtual HRESULT	RegisterProperty (_in LPCWSTR pszPropertyName, _in _notnull PEventStringCallback) PURE
virtual HANDLE	RegisterSystemMalfunction (_in REFGUID guidMalfunction, _in

	LPCWSTR pszType, __in LPCWSTR pszBaseName, __in LPCWSTR pszInstanceName, __in int nSubIndex) PURE
virtual float	GetSystemHealth (HANDLE hSystem) const PURE
virtual HRESULT	DecrementHealthPoints (__in float fDamagePoints) PURE
virtual float	GetHealthPoints () const PURE
virtual void	SetHealthPoints (float fHealthPoints) PURE
virtual HRESULT	GetSurfaceInformation (__out SurfaceInfoV400 &SurfaceInfo, __in_opt const FXYZ *pvOffsetFeet) PURE
virtual HRESULT	GetSurfaceElevation (__out float &fElevationFeet, __in_opt const FXYZ *pvOffsetFeet) PURE
virtual HRESULT	GetBathymetryElevation (__out float &fDepthFeet, __in_opt const FXYZ *pvOffsetFeet) PURE
virtual HRESULT	GetWeatherInformation (__out WeatherInfoV400 &WeatherInfo) PURE
virtual float	GetMagneticVariation () const PURE
virtual HRESULT	VisualEffectOn (__in __notnull LPCWSTR pszEffectName, __in_opt const FXYZ *pvOffsetFeet, __out void **ppEffect) PURE
virtual HRESULT	VisualEffectOff (__in __notnull void *pEffect) PURE
virtual HRESULT	TriggerSound (__in __notnull LPCWSTR pszName, BOOL bOn) PURE
virtual HRESULT	TriggerContactSound (__in __notnull LPCWSTR pszName, __in const FXYZ *pvOffset, float flImpactSpeed) PURE
virtual HRESULT	StopSound (__in __notnull LPCWSTR pszName) PURE
virtual HRESULT	LoadServiceConstantData (__in REFGUID guidService) PURE
virtual HRESULT	UnloadServiceConstantData (__in REFGUID guidService) PURE
virtual HRESULT	CreateServiceInstance (__in REFGUID guidService) PURE
virtual HRESULT	DestroyServiceInstance (__in REFGUID guidService) PURE
virtual HRESULT	UpdateServiceInstance (__in REFGUID guidService, double dDeltaT) PURE
virtual HRESULT	Title (__out LPWSTR pszCfgTitle, __in unsigned int uLength) const PURE
virtual HRESULT	GetCfgDir (__out LPWSTR pszCfgDir, __in unsigned int uLength) const PURE
virtual HRESULT	GetCfgFilePath (__out LPWSTR pszCfgFile, __in unsigned int uLength) const PURE
virtual HRESULT	GetCfgSectionName (__out LPWSTR pszCfgFile, __in unsigned int uLength) const PURE
virtual HRESULT	Destroy () PURE
virtual HRESULT	CheckCollision (__in float fRadiusFeet, __out COLLISIONTYPE &eCollision, __out IUnknown **ppUnkHitObject) const PURE
virtual HRESULT	CheckCollision (__in float fRadiusFeet, __in const DXYZ *pdxyzPoints, __in INT32 uPointCount, __out COLLISIONTYPE &eCollision, __out IUnknown **ppUnkHitObject) const PURE
virtual NET_MODE_TYPE	GetNetworkMode () const PURE
virtual HRESULT	AttachObject (__in const DXYZ &vOffsetFeetParent, __in const DXYZ &vOffsetRadiansParent, __in UINT idChild, __in const DXYZ &vOffsetFeetChild, __in const DXYZ &vOffsetRadiansChild) PURE
virtual HRESULT	AttachObject (__in LPCSTR pszAttachPointName, __in const DXYZ &vOffsetRadiansParent, __in UINT idChild, __in const DXYZ &vOffsetFeetChild, __in const DXYZ &vOffsetRadiansChild) PURE
virtual HRESULT	DetachObject (__in UINT idChild) PURE
virtual HRESULT	GetCategoryName (__out LPWSTR pszCategoryName, __in unsigned int uLength) const PURE
virtual HRESULT	GetCategoryId (__out GUID &guidCategory) const PURE
virtual UINT	GetDamageState () const PURE
virtual void	SetDamageState (UINT uDamageState) PURE
virtual HRESULT	GetBoundingBox (__out DXYZ &dxyzMin, __out DXYZ &dxyzMax) const PURE
virtual HRESULT	GetCrashTreeBox (__in UINT index, __out DXYZ &dxyzMin, __out DXYZ &dxyzMax) const PURE
virtual UINT	GetCrashTreeBoxCount () const PURE

Member Function Documentation

◆ **AttachObject()** [1 / 2]

```
virtual HRESULT AttachObject (__in const DXYZ & vOffsetFeetParent,
                             __in const DXYZ & vOffsetRadiansParent,
                             __in UINT          idChild,
                             __in const DXYZ & vOffsetFeetChild,
                             __in const DXYZ & vOffsetRadiansChild
                           )
```

private **virtual**

Attaches the given object via offsets.

Parameters

vOffsetFeetParent The offset in feet from the parent model center.
vOffsetRadiansParent The orientation offset in radians from the parent model center.
idChild The child object id to be attached to the parent.
vOffsetFeetChild The offset in feet from the parent attach point.
vOffsetRadiansChild The orientation offset in radians from the parent attach point.

Returns

S_OK if the objects were successfully attached, E_FAIL otherwise.

◆ **AttachObject()** [2/2]

```
virtual HRESULT AttachObject ( __in LPCSTR      pszAttachPointName,
                             __in const DXYZ & vOffsetRadiansParent,
                             __in UINT          idChild,
                             __in const DXYZ & vOffsetFeetChild,
                             __in const DXYZ & vOffsetRadiansChild
                           )
```

private virtual

Attaches the given object via attach point name and offsets.

Parameters

pszAttachPointName The name of the parent attach point.
vOffsetRadiansParent The orientation offset in radians from the parent attach point.
idChild The child object id to be attached to the parent attach point.
vOffsetFeetChild The offset in feet from the parent attach point.
vOffsetRadiansChild The orientation offset in radians from the parent attach point.

Returns

S_OK if the objects were successfully attached, E_FAIL otherwise.

◆ **CheckCollision()** [1/2]

```
virtual HRESULT CheckCollision ( __in float           fRadiusFeet,
                                __out COLLISIONTYPE & eCollision,
                                __out IUnknown **    ppUnkHitObject
                              )
```

private virtual

Checks if this object's center point is colliding with a building or another SimObject.

Parameters

fRadiusFeet The radius around the object to check for collisions (feet).
eCollision The resulting collision type.
ppUnkHitObject The object the collision happened with.

Returns

S_OK if successful, E_FAIL otherwise. Note: A return of S_OK does not mean there is a collision, only that the query operation encountered no errors. The eCollision should be checked for a positive collision, and ppUnkHitObject for whether it involved an (IUnknown) object

◆ **CheckCollision()** [2/2]

```
virtual HRESULT CheckCollision ( __in float           fRadiusFeet,
                                __in const DXYZ *   pdxyzPoints,
                                __in UINT32         uPointCount,
                                __out COLLISIONTYPE & eCollision,
                                __out IUnknown **    ppUnkHitObject
                              )
```

private virtual

Checks if any of the given points are colliding with a building or another SimObject.

Parameters

fRadiusFeet The radius around the object to check for collisions (feet).
pdxyzPoints Body relative offset points used for collision detection (feet).
uPointCount The number of points in pdxyzPoints.
eCollision The resulting collision type.
ppUnkHitObject The object the collision happened with.

Returns

S_OK if successful, E_FAIL otherwise. Note: A return of S_OK does not mean there is a collision, only that the query operation encountered no errors. The eCollision should be checked for a positive collision, and ppUnkHitObject for whether it involved an (IUnknown) object

◆ **CreateServiceInstance()**

```
virtual HRESULT CreateServiceInstance ( __in REFGUID guidService )
```

private virtual

Invokes the instantiation of the service, based on the loaded constant data. This should be called from your SimObject's LoadDynamicData();

◆ **DecrementHealthPoints()**

virtual HRESULT DecrementHealthPoints (__in float fDamagePoints) **private** **virtual**

Apply damage points. Positive points passed in will be decremented from current health points, to a limit of zero.

◆ **Destroy()**

virtual HRESULT Destroy () **private** **virtual**

Destroy self. This will not be immediate, so it can be called from within itself. It will be destroyed as soon as the current simulation finishes.

◆ **DestroyServiceInstance()**

virtual HRESULT DestroyServiceInstance (__in REFGUID guidService) **private** **virtual**

Causes Prepar3D to destroy the instance of the service. This should be called from your SimObject's Deinit();

◆ **DetachObject()**

virtual HRESULT DetachObject (__in UINT idChild) **private** **virtual**

Detaches the given object.

Parameters

idChild The child object id to be detached from the parent.

Returns

S_OK if the object was successfully detached, S_FALSE if the object to be removed was not attached to the parent, E_FAIL otherwise.

◆ **GetBathymetryElevation()**

virtual HRESULT GetBathymetryElevation (__out float & fDepthFeet,
 __in_opt const FXYZ * pvOffsetFeet
)

private **virtual**

Provides current depth for the requested offset from the model center.

Parameters

fDepthFeet Reference to the depth variable (Feet)

pvOffsetFeet The offset from model enter. A value of NULL will use the model's center

◆ **GetBoundingBox()**

virtual HRESULT GetBoundingBox (__out DXYZ & dxyzMin,
 __out DXYZ & dxyzMax
)

const **private** **virtual**

Gets the bounding box of the object.

Parameters

dxyzMin The minimum x, y, z values of the box in feet.

dxyzMax The maximum x, y, z values of the box in feet.

Returns

E_FAIL on failure, S_OK on success.

◆ **GetCategoryId()**

virtual HRESULT GetCategoryId (__out GUID & guidCategory) **const** **private** **virtual**

Returns the given simobject's category ID.

Parameters

guidCategory The category ID of the object

Returns

S_OK if the object's category ID was returned

◆ **GetCategoryName()**

virtual HRESULT GetCategoryName (__out LPWSTR pszCategoryName,

```
    __in unsigned int uLength  
    ) const private virtual
```

Returns the given simobject's category name.

Parameters

pszCategoryName The buffer to store the category name of the object
uLength The length of the buffer to store the category name in characters

Returns

S_OK if the object's category name was returned

◆ **GetCfgDir()**

```
virtual HRESULT GetCfgDir ( __out LPWSTR pszCfgDir,  
    __in unsigned int uLength  
    ) const private virtual
```

Returns a fully qualified path to the object's content path. This is generally the folder where the sim.cfg (or aircraft.cfg) lives.

◆ **GetCfgFilePath()**

```
virtual HRESULT GetCfgFilePath ( __out LPWSTR pszCfgFile,  
    __in unsigned int uLength  
    ) const private virtual
```

Returns a fully qualified path to the sim.cfg file.

◆ **GetCfgSectionName()**

```
virtual HRESULT GetCfgSectionName ( __out LPWSTR pszCfgFile,  
    __in unsigned int uLength  
    ) const private virtual
```

Returns the relevant section name in the sim.cfg. e.g. [fltsim.1].

◆ **GetCrashTreeBox()**

```
virtual HRESULT GetCrashTreeBox ( __in UINT index,  
    __out DXYZ & dxyzMin,  
    __out DXYZ & dxyzMax  
    ) const private virtual
```

Gets the crash tree boxes of the object via index.

Parameters

index The index of the crash tree box.
dxyzMin The minimum x, y, z values of the box in feet.
dxyzMax The maximum x, y, z values of the box in feet.

Returns

E_FAIL on failure, S_OK on success.

◆ **GetCrashTreeBoxCount()**

```
virtual UINT GetCrashTreeBoxCount ( ) const private virtual
```

Gets the crash tree box count.

Returns

The number of crash tree boxes.

Remarks

Returns 0 when no crash tree is found.

◆ **GetDamageState()**

```
virtual UINT GetDamageState ( ) const private virtual
```

Gets the damage state

Returns

The damage state

Remarks

0 = No Damage, 1 = Light, 2 = Moderate, 3 = Destroyed, 4-n = User Defined

◆ GetHealthPoints()

virtual float GetHealthPoints() const **private** **virtual**

Retrieves the current health of the object.

◆ GetId()

virtual **UINT** GetId() const **private** **virtual**

The ID of this object: NOTE: "0" is an invalid id

◆ GetMagneticVariation()

virtual float GetMagneticVariation() const **private** **virtual**

Returns the object's current magnetic variation in radians. A positive is value is "east".

◆ GetMainMinMaxSimRates()

virtual **HRESULT** GetMainMinMaxSimRates(__out float & fMinHz,
__out float & fMaxHz
) const **private** **virtual**

Provides the minimum and maximum main simulation rate (Hz). Typically the world position update rate. An ISimObject implementation will be called by this if it exists.

◆ GetMissionId()

virtual **HRESULT** GetMissionId(__out **GUID** & guid) const **private** **virtual**

The guid ID of the object defined in an object file. NOTE: If the object is not spawned by a scenario, the return will be E_FAIL and the ID will be GUID_NULL.

◆ GetMode()

virtual **int** GetMode() const **private** **virtual**

Returns bitwise flags for the current modes of the SimObject.

◆ GetNetworkMode()

virtual **NET_MODE_TYPE** GetNetworkMode() const **private** **virtual**

Returns the current network mode for this object.

◆ GetObjectGroupAssociationId()

virtual **UINT** GetObjectGroupAssociationId() const **private** **virtual**

Group Association ID can be used to set/get IDs for Friend/Foe or other types of groupings.
Note: Group association is arbitrary. It could be used for things like alliances or squadrons. Default is 0, which signifies a neutral grouping.

◆ GetPosition()

virtual **HRESULT** GetPosition(__out **DXYZ** & vLonAltLat,
__out **DXYZ** & vPHB,
__out **DXYZ** & vLonAltLatVel,
__out **DXYZ** & vPHBVel
) const **private** **virtual**

Gets the current world relative position and velocity from the Prepar3D-side SimObject. This will provide the valid state upon initialization, as well as when another system such as a UI element or slew changes the position.

Parameters

vLonAltLat Longitude, altitude, latitude (radians)

vPHB Pitch, heading, bank (radians)

vLonAltLatVel Longitude, altitude, latitude velocity (feet / second)

vPHBVel Pitch, heading, bank velocity (radians / second)

◆ GetProperty() [1/11]

```
virtual HRESULT GetProperty ( __in int          iPropertyNameCode,
                            __in int          iUnitCode,
                            __out double &   dProperty,
                            __in int          index = 0
                           )                const    private virtual
```

Get Property - Doubles

◆ GetProperty() [2/11]

```
virtual HRESULT GetProperty ( __in LPCWSTR  pszPropertyName,
                            __in int          iUnitCode,
                            __out double &   dProperty,
                            __in int          index = 0
                           )                const    private virtual
```

Get Property - Doubles

◆ GetProperty() [3/11]

```
virtual HRESULT GetProperty ( __in LPCWSTR  pszPropertyName,
                            __in LPCWSTR  pszUnitCode,
                            __out double &   dProperty,
                            __in int          index = 0
                           )                const    private virtual
```

Get Property - Doubles

◆ GetProperty() [4/11]

```
virtual HRESULT GetProperty ( __in int          iPropertyNameCode,
                            __in int          iUnitCode,
                            __out DXYZ &    dProperty,
                            __in int          index = 0
                           )                const    private virtual
```

Get Property - Vectors

◆ GetProperty() [5/11]

```
virtual HRESULT GetProperty ( __in LPCWSTR  pszPropertyName,
                            __in int          iUnitCode,
                            __out DXYZ &    dProperty,
                            __in int          index = 0
                           )                const    private virtual
```

Get Property - Vectors

◆ GetProperty() [6/11]

```
virtual HRESULT GetProperty ( __in LPCWSTR  pszPropertyName,
                            __in LPCWSTR  pszUnitCode,
                            __out DXYZ &    dProperty,
                            __in int          index = 0
                           )                const    private virtual
```

Get Property - Vectors

◆ GetProperty() [7/11]

```
virtual HRESULT GetProperty ( __in int          iPropertyNameCode,
                            __out LPWSTR    pszProperty,
                            __in UINT        uLength,
                            __in int          index = 0
                           )                const    private virtual
```

Get Property - Strings

◆ GetProperty() [8/11]

```
virtual HRESULT GetProperty ( __in LPCWSTR pszPropertyName,
                            __out LPWSTR pszProperty,
                            __in UINT uLength,
                            __in int index = 0
                        )

```

const private virtual

Get Property - Strings

◆ GetProperty() [9/11]

```
virtual HRESULT GetProperty ( __in int iPropertyNameCode,
                            __in LPCWSTR pszSecondarySubstring,
                            __in int iUnitCode,
                            __out double & dProperty,
                            __in int index = 0
                        )

```

const private virtual

Get Property - Doubles (with secondary substring input)

◆ GetProperty() [10/11]

```
virtual HRESULT GetProperty ( __in LPCWSTR pszPropertyName,
                            __in LPCWSTR pszSecondarySubstring,
                            __in int iUnitCode,
                            __out double & dProperty,
                            __in int index = 0
                        )

```

const private virtual

Get Property - Doubles (with secondary substring input)

◆ GetProperty() [11/11]

```
virtual HRESULT GetProperty ( __in LPCWSTR pszPropertyName,
                            __in LPCWSTR pszSecondarySubstring,
                            __in LPCWSTR pszUnitCode,
                            __out double & dProperty,
                            __in int index = 0
                        )

```

const private virtual

Get Property - Doubles (with secondary substring input)

◆ GetPropertyCodeAndIndex()

```
virtual HRESULT GetPropertyCodeAndIndex ( __in PROPERTY_TYPE eType,
                                         __in LPCWSTR pszPropertyName,
                                         __out int & iPropertyNameCode,
                                         __inout int & iIndex
                                       )

```

const private virtual

Get Properties

◆ GetSurfaceElevation()

```
virtual HRESULT GetSurfaceElevation ( __out float & fElevationFeet,
                                      __in_opt const FXYZ * pvOffsetFeet
                                    )

```

private virtual

Provides current surface elevation (above Mean Sea Level) for the requested offset from the model center. This will be more efficient than GetSurfaceInformation when only the elevation is needed. A return value of E_FAIL means that Prepar3D's terrain system failed to process the request properly. This could happen if it is not initialized fully.

Parameters

fElevationFeet Reference to the elevation variable (Feet)

pvOffsetFeet The offset from model center. A value of NULL will use the model's center

◆ GetSurfaceInformation()

```
virtual HRESULT GetSurfaceInformation ( __out SurfaceInfoV400 & SurfaceInfo,
                                         __in_opt const FXYZ * pvOffsetFeet
                                       )

```

private virtual

Provides current surface information for the requested offset from the model center. See the **ISimObject.h** for the definition of the SurfaceInfo data structure. A return value of E_FAIL means that

Prepar3D's terrain system failed to process the request properly. This could happen if it is not initialized fully.

Parameters

SurfaceInfo Reference to local SurfaceInfo data structure
pvOffsetFeet The offset from model center. A value of NULL will use the model's center

◆ GetSystemHealth()

virtual float GetSystemHealth (HANDLE hSystem) const [private] [virtual]

Returns the health percentage (0.0 - 1.0) for a given malfunction.

◆ GetTitle()

virtual HRESULT GetTitle (__out LPWSTR pszCfgTitle,
 __in unsigned int ulLength
) const [private] [virtual]

Returns the unique string that identifies this object.

◆ GetWeatherInformation()

virtual HRESULT GetWeatherInformation (__out WeatherInfoV400 & WeatherInfo) [private] [virtual]

Provides current weather information for the object's current position. See the **ISimObject.h** for the definition of the WeatherInfo data structure. A return value of E_FAIL means that Prepar3D's weather system failed to process the request properly. This could happen if it is not initialized fully.

Parameters

WeatherInfo Reference to local WeatherInfo data structure

◆ InitPosition()

virtual HRESULT InitPosition (__in const DXYZ * pvLonAltLat,
 __in const DXYZ * pvPHB,
 __in const DXYZ * pvLonAltLatVel,
 __in const DXYZ * pvPHBVel,
 __in BOOL bSetOnGround
) [private] [virtual]

Sets the current world relative position and velocity to the Prepar3D-side SimObject. All parameters are optional. Any parameter set to NULL will be ignored, and current object values will be retained.

Parameters

pvLonAltLat Longitude, altitude, latitude (radians)
pvPHB Pitch, heading, bank (radians)
pvLonAltLatVel Longitude, altitude, latitude velocity (feet / second)
pvPHBVel Pitch, heading, bank velocity (radians / second)
bSetOnGround Flag indicating if the object is to be set on the ground, in which case the on-ground height and pitch attitude will be set.

◆ InObjectFoeList()

virtual BOOL InObjectFoeList (UINT id) const [private] [virtual]

Group ObjectFoeList Hosts a list of ID's that are considered foe's to the current entity

◆ InObjectFriendList()

virtual BOOL InObjectFriendList (UINT id) const [private] [virtual]

Group ObjectFoeList Hosts a list of ID's that are considered friends's to the current entity

◆ IsOnGround()

virtual BOOL IsOnGround () const [private] [virtual]

IsOnGround Returns the on-ground flag value currently in the core base object. This can be useful for determining if the object has been placed on the ground through a non-simulated means such as the UI.

◆ IsUser()

virtual BOOL IsUser() const [private] [virtual]

Returns if the object is the user or not.

◆ LoadServiceConstantData()

virtual HRESULT LoadServiceConstantData(__in REFGUID guidService) [private] [virtual]

Invokes the loading of the relevant constant data. This should be called from your SimObject's LoadConstantData();

◆ RegisterProperty() [1 / 6]

virtual HRESULT RegisterProperty(__in LPCWSTR pszPropertyName,
__in LPCWSTR pszPropertyBaseUnits,
__in __notnull PPropertyCallback pcbProperty
)

[private] [virtual]

Property "simvar" registration. For this specific object, associates: property string name, units, and callback pointer (defined above) This differs from the RegisterProperty method in the ISimObjectManager interface which registers a property that exists for all instances associated with the given guid. Properties registered here, as well as their ID, exist only for the lifetime of this object. Otherwise they function the same when used for gauges, animations, and missions. Properties must also be declared in Properties.xml for the object. See the PDK samples.

Note

Input: Double

◆ RegisterProperty() [2 / 6]

virtual HRESULT RegisterProperty(__in LPCWSTR pszPropertyName,
__in LPCWSTR pszPropertyBaseUnits,
__in __notnull PEventCallback pcbEvent,
__in EVENTTYPE eType
)

[private] [virtual]

Property "event" registration. For this specific object, associates: property string name, units, and callback pointer (defined above). This differs from the RegisterProperty method in the ISimObjectManager interface which registers a property that exists for all instances associated with the given guid. Properties registered here, as well as their ID, exist only for the lifetime of this object. Otherwise they function the same when used for gauges, animations, and missions. Properties must also be declared in Properties.xml for the object. See the PDK samples.

Note

Input: Event

◆ RegisterProperty() [3 / 6]

virtual
HRESULT
RegisterProperty(__in LPCWSTR pszPropertyName,
__in LPCWSTR pszPropertyBaseUnits,
__in __notnull PPropertyVectorCallback pcbProperty
)

[private] [virtual]

Property "simvar" registration. For this specific object, associates: property string name, units, and callback pointer (defined above). This differs from the RegisterProperty method in the ISimObjectManager interface which registers a property that exists for all instances associated with the given guid. Properties registered here, as well as their ID, exist only for the lifetime of this object. Otherwise they function the same when used for gauges, animations, and missions. Properties must also be declared in Properties.xml for the object. See the PDK samples.

Note

Input: Vector (DXYZ)

◆ RegisterProperty() [4 / 6]

virtual HRESULT RegisterProperty(__in LPCWSTR pszPropertyName,
__in LPCWSTR pszPropertyBaseUnits,
__in __notnull PEventVectorCallback
)

[private] [virtual]

Property "event" registration. For this specific object, associates: property string name, units, and callback pointer (defined above). This differs from the RegisterProperty method in the ISimObjectManager interface which registers a property that exists for all instances associated with the given guid. Properties registered here, as well as their ID, exist only for the lifetime of this object. Otherwise they function the same when used for gauges, animations, and missions. Properties must also be declared in Properties.xml for the object. See the PDK samples.

Note

Input: Event vector

◆ RegisterProperty() [5 / 6]

```
virtual HRESULT RegisterProperty( __in LPCWSTR pszPropertyName,
                                 __in __notnull PPropertyStringCallback pcbProperty
                               )
```

private virtual

Property "simvar" registration. For this specific object, associates: property string name, units, and callback pointer (defined above). This differs from the RegisterProperty method in the ISimObjectManager interface which registers a property that exists for all instances associated with the given guid. Properties registered here, as well as their ID, exist only for the lifetime of this object. Otherwise they function the same when used for gauges, animations, and missions. Properties must also be declared in Properties.xml for the object. See the PDK samples.

Note

Input: String

◆ RegisterProperty() [6 / 6]

```
virtual HRESULT RegisterProperty( __in LPCWSTR pszPropertyName,
                                 __in __notnull PEventStringCallback
                               )
```

private virtual

Property "event" registration. For this specific object, associates: property string name, units, and callback pointer (defined above). This differs from the RegisterProperty method in the ISimObjectManager interface which registers a property that exists for all instances associated with the given guid. Properties registered here, as well as their ID, exist only for the lifetime of this object. Otherwise they function the same when used for gauges, animations, and missions. Properties must also be declared in Properties.xml for the object. See the PDK samples.

Note

Input: Event string

◆ RegisterService()

```
virtual HRESULT RegisterService( __in REFGUID guidService,
                                __in __notnull IUnknown * punkService
                              )
```

private virtual

Registers a service that can be queried for on this object. A service should be an IUnknown-derived object and registered with a unique GUID.

Parameters**guidService** Unique GUID to identify this service.**punkService** Reference to an instance of this service.**◆ RegisterSimulation() [1 / 2]**

```
virtual HRESULT RegisterSimulation( __in __notnull ISimulation * pSimulation,
                                    float fRateHz
                                  )
```

private virtual

Registers an ISimulation callback for real-time updates (discussed in Creating Behaviors.) ISimulation registration will be locked after the ISimObject Init() function has been called. All ISimulation objects must be registered before this point.

Parameters**pSimulation** Address of simulation system**fRateHz** Specified iteration rate**◆ RegisterSimulation() [2 / 2]**

```
virtual HRESULT RegisterSimulation( __in __notnull ISimulation * pSimulation,
                                    float fMinRateHz,
                                    float fMaxRateHz
                                  )
```

private virtual

Registers an ISimulation callback for real-time updates (discussed in Creating Behaviors.) ISimulation registration will be locked after the ISimObject Init() function has been called. All ISimulation objects must be registered before this point.

Parameters**pSimulation** Address of simulation system**fMinRateHz** Specified minimum iteration rate**fMaxRateHz** Specified maximum iteration rate

◆ RegisterSystemMalfunction()

```
virtual HANDLE RegisterSystemMalfunction( __in REFGUID guidMalfunction,
                                         __in LPCWSTR pszType,
                                         __in LPCWSTR pszBaseName,
                                         __in LPCWSTR pszInstanceName,
                                         __in int         nSubIndex
                                         )
```

private virtual

Registers a specific malfunction that can be set through the UI, scenarios, or missions.

Parameters

guidMalfunction Unique malfunction ID
pszType UI Type. Choices: Instruments, Systems, Radios, Engines, Controls, Structural, Miscellaneous
pszBaseName Name used for mission file reference. Should be generic (no index), such as "Engine"
pszInstanceName Specific malfunction name for the UI, such as "Total Failure Engine 1"
nSubIndex Sub-index. For example, engine 0, 1, etc...

◆ RotateBodyToWorld()

```
virtual HRESULT RotateBodyToWorld( __in const DXYZ & vBody,
                                   __out DXYZ &      vWorld
                                   )
```

const private virtual

Rotates a vector from the body frame of reference to the world frame of reference.

◆ RotateWorldToBody()

```
virtual HRESULT RotateWorldToBody( __in const DXYZ & vWorld,
                                   __out DXYZ &      vBody
                                   )
```

const private virtual

Rotates a vector from the world frame of reference to the body frame of reference.

◆ SetCrashMode()

```
virtual HRESULT SetCrashMode( double dDeltaT )
```

private virtual

Should be called when it is desired to put Prepar3D into "crash" mode. By default, the application will go through its crash cycle and reset. It is the developer's responsibility to program the behavior of the object when crash in crash mode.

◆ SetDamageState()

```
virtual void SetDamageState( UINT uDamageState )
```

private virtual

Sets the damage state

Parameters

eDamageState The damage state

Remarks

0 = No Damage, 1 = Light, 2 = Moderate, 3 = Destroyed, 4-n = User Defined

◆ SetHealthPoints()

```
virtual void SetHealthPoints( float fHealthPoints )
```

private virtual

Sets current health of the object.

◆ SetMainMinMaxSimRates()

```
virtual HRESULT SetMainMinMaxSimRates( __in float fMinHz,
                                         __in float fMaxHz
                                         )
```

private virtual

Sets minimum and maximum main simulation rate (Hz). Typically the world position update rate. This is typically called if there is a desire to synchronize the rates of two or more objects. For example an aircraft and an aircraft carrier, or to prevent perceived jitter when viewing an object from a camera attached to another object. An ISimObject implementation will be called by this if it exists.

◆ SetObjectFoeList()

```
virtual void SetObjectFoeList ( UINT * uEnteredFoeID,
                               UINT size
                           )
```

private virtual

Group Association ID can be used to set/get IDs for Friend/Foe or other types of groupings.

◆ SetObjectFriendList()

```
virtual void SetObjectFriendList ( UINT * uEnteredFriendID,
                                   UINT size
                               )
```

private virtual

Group Association ID can be used to set/get IDs for Friend/Foe or other types of groupings.

◆ SetObjectGroupAssociationId()

```
virtual void SetObjectGroupAssociationId ( UINT uAssociationId )
```

private virtual

Group Association ID can be used to set/get IDs for Friend/Foe or other types of groupings.

Note: Group association is arbitrary. It could be used for things like alliances or squadrons. Default is 0, which signifies a neutral grouping.

◆ SetPosition()

```
virtual HRESULT SetPosition ( __in const DXYZ & vLonAltLat,
                             __in const DXYZ & vPHB,
                             __in const DXYZ & vLonAltLatVel,
                             __in const DXYZ & vPHBVel,
                             __in BOOL     blsOnGround,
                             __in double   dDeltaT
                           )
```

private virtual

Sets the current world relative position and velocity to the Prepar3D-side SimObject.

Parameters

vLonAltLat Longitude, altitude, latitude (radians)
vPHB Pitch, heading, bank (radians)
vLonAltLatVel Longitude, altitude, latitude velocity (feet / second)
vPHBVel Pitch, heading, bank velocity (radians / second)
blsOnGround Flag indicating if the object is on the ground. This is important during terrain updates.
dDeltaT The time between object updates used to track how much time has accumulated between camera frames

◆ StopSound()

```
virtual HRESULT StopSound ( __in __notnull LPCWSTR pszName )
```

private virtual

Stops a sound configured in the object's sound.cfg. This function will stop a looping or a one shot sound.

Parameters

pszName Sound reference name from Sound.cfg

◆ TriggerContactSound()

```
virtual HRESULT TriggerContactSound ( __in __notnull LPCWSTR pszName,
                                       __in const FXYZ * pvOffset,
                                       float           flImpactSpeed
                                     )
```

private virtual

Triggers a sound specifically for a ground contact point.

Parameters

pszName Sound reference name from Sound.cfg
pvOffset The offset from model center. A value of NULL will use the model's center
flImpactSpeed Speed used by sound system to scale sound

◆ TriggerProperty() [1/11]

```
virtual HRESULT TriggerProperty ( __in int    iPropertyName,
                                 __in int    iUnitCode,
                                 __in double dData,
                                 __in int    index
                               )
```

const

private virtual

Numeric trigger

◆ TriggerProperty() [2/11]

```
virtual HRESULT TriggerProperty( __in LPCWSTR pszPropertyName,  
                                __in int iUnitCode,  
                                __in double dData,  
                                __in int index  
                                ) const
```

[private] [virtual]

Numeric trigger

◆ TriggerProperty() [3/11]

```
virtual HRESULT TriggerProperty( __in LPCWSTR pszPropertyName,  
                                __in LPCWSTR pszUnitCode,  
                                __in double dData,  
                                __in int index  
                                ) const
```

[private] [virtual]

Numeric trigger

◆ TriggerProperty() [4/11]

```
virtual HRESULT TriggerProperty( __in int iPropertyCode,  
                                __in int iUnitCode,  
                                __in const DXYZ & vData,  
                                __in int index  
                                ) const
```

[private] [virtual]

Vector trigger

◆ TriggerProperty() [5/11]

```
virtual HRESULT TriggerProperty( __in LPCWSTR pszPropertyName,  
                                __in int iUnitCode,  
                                __in const DXYZ & vData,  
                                __in int index  
                                ) const
```

[private] [virtual]

Vector trigger

◆ TriggerProperty() [6/11]

```
virtual HRESULT TriggerProperty( __in LPCWSTR pszPropertyName,  
                                __in LPCWSTR pszUnitCode,  
                                __in const DXYZ & vData,  
                                __in int index  
                                ) const
```

[private] [virtual]

Vector trigger

◆ TriggerProperty() [7/11]

```
virtual HRESULT TriggerProperty( __in int iPropertyCode,  
                                __in LPCWSTR pszData,  
                                __in int index  
                                ) const
```

[private] [virtual]

String trigger

◆ TriggerProperty() [8/11]

```
virtual HRESULT TriggerProperty( __in LPCWSTR pszPropertyName,  
                                __in LPCWSTR pszData,  
                                __in int index  
                                ) const
```

[private] [virtual]

String trigger

◆ TriggerProperty() [9/11]

```
virtual HRESULT TriggerProperty ( __in int iPropertyNameCode,
                                __in LPCWSTR pszSecondarySubstring,
                                __in int iUnitCode,
                                __in double dData,
                                __in int index
                            )
) const private virtual
```

Trigger - Doubles (with secondary substring input)

◆ TriggerProperty() [10/11]

```
virtual HRESULT TriggerProperty ( __in LPCWSTR pszPropertyName,
                                __in LPCWSTR pszSecondarySubstring,
                                __in int iUnitCode,
                                __in double dData,
                                __in int index
                            )
) const private virtual
```

Trigger - Doubles (with secondary substring input)

◆ TriggerProperty() [11/11]

```
virtual HRESULT TriggerProperty ( __in LPCWSTR pszPropertyName,
                                __in LPCWSTR pszSecondarySubstring,
                                __in LPCWSTR pszUnitCode,
                                __in double dData,
                                __in int index
                            )
) const private virtual
```

Trigger - Doubles (with secondary substring input)

◆ TriggerSound()

```
virtual HRESULT TriggerSound ( __in __notnull LPCWSTR pszName,
                             BOOL bOn
                           )
) private virtual
```

Triggers a sound configured in the object's sound.cfg.

Parameters

pszName Sound reference name from Sound.cfg
bOn Turns on/off a looping sound. This value has no effect on one shot sounds.

Remarks

To turn off a one shot sound, use the StopSound function.

◆ UnloadServiceConstantData()

```
virtual HRESULT UnloadServiceConstantData ( __in REFGUID guidService ) private virtual
```

Causes Prepar3D to unload the relevant constant data. This should be called from your SimObject's UnLoadConstantData();

◆ UnregisterService()

```
virtual HRESULT UnregisterService ( __in REFGUID guidService ) private virtual
```

Removes a service that has been register with [RegisterService\(\)](#).

Parameters

guidService Unique GUID to identify this service.

◆ UpdateServiceInstance()

```
virtual HRESULT UpdateServiceInstance ( __in REFGUID guidService,
                                       double dDeltaT
                                     )
) private virtual
```

The real-time update of the service. Your SimObject is responsible for calling it with an accurate delta time.

◆ VisualEffectOff()

```
virtual HRESULT VisualEffectOff ( __in __notnull void * pEffect ) private virtual
```

Turns a visual effect off.

Parameters

pEffect Reference pointer obtained in out parameter of [VisualEffectOn\(\)](#)

◆ [VisualEffectOn\(\)](#)

```
virtual HRESULT VisualEffectOn ( __in __notnull LPCWSTR pszEffectName,
                                __in_opt const FXYZ * pvOffsetFeet,
                                __out void ** ppEffect
                            )
```

private virtual

Turns a visual effect on. The out parameter allows you to hold a reference to a visual effect to subsequently turn off.

Parameters

pszEffectName File name for requested visual effect
pvOffsetFeet The offset from model center. A value of NULL will use the model's center
ppEffect Reference pointer for turning the visual effect off with [VisualEffectOff\(\)](#)

◆ [P3D::ISubSystemFactoryV500](#)

class P3D::ISubSystemFactoryV500

Factory class interface for creating supplemental subsystems on an existing simobject implementation

There are two separate methods for creating a system with registered factory.

- [Create\(\)](#) - This creates the object and registers it directly with the simulation manager for real-time callbacks.
- [CreateEx\(\)](#) - Returns an IUnknown reference to the created object, in which the caller will be responsible for all real-time callbacks.

[Create\(\)](#) will be called first, and [CreateEx\(\)](#) will only be called if the former does not succeed.

Inherits ISubSystemFactoryV440.

Private Member Functions

```
virtual HRESULT Create ( __in IBaseObjectV400 * pBaseObject, __in LPCWSTR
                        pszSecondaryData) PURE
```

```
virtual HRESULT CreateEx ( __in IBaseObjectV400 * pBaseObject, __in LPCWSTR
                           pszSecondaryData, __out IUnknown **punkSystem) PURE
```

Member Function Documentation

◆ [Create\(\)](#)

```
virtual HRESULT Create ( __in IBaseObjectV400 * pBaseObject,
                        __in LPCWSTR           pszSecondaryData
                    )
```

private virtual

Creates a new subsystem during object loading

Parameters

pBaseObject The object on which the subsystem is being attached
pszSecondaryData (optional) This allows unique subsystems to be specified using the same factory.

Remarks

The SubSystemFactory should be registered through the [IPdK](#) interface at DLL load time.

Supplemental subsystems can be specified in the aircraft.cfg/sim.cfg file. For example:

[SupplementalSystems]

System.0 = {bc95b363-1d22-42aa-82b1-f10905b22c40}, Engine

System.1 = {bc95b363-1d22-42aa-82b1-f10905b22c40}, Propeller

where the guid is the registered Service ID (SID)

◆ [CreateEx\(\)](#)

```
virtual HRESULT CreateEx ( __in IBaseObjectV400 * pBaseObject,
                           __in LPCWSTR           pszSecondaryData,
                           __out IUnknown **       punkSystem
                       )
```

private virtual

Creates a new subsystem during object loading

Parameters

pBaseObject The object on which the subsystem is being attached

pszSecondaryData (optional) This allows unique subsystems to be specified using the same factory.

punkSystem Returns reference to the object created.

Remarks

The SubSystemFactory should be registered through the **IPdk** interface at DLL load time.

Supplemental subsystems can be specified in the aircraft.cfg/sim.cfg file. For example:

[SupplementalSystems]

System.0 = {bc95b363-1d22-42aa-82b1-f10905b22c40}, Engine

System.1 = {bc95b363-1d22-42aa-82b1-f10905b22c40}, Propeller

where the guid is the registered Service ID (SID)

◆ P3D::WorldConstants

class P3D::WorldConstants

Constant values describing the Earth atmosphere and geometry

Note

While this is accessed through IBaseObject, these values will be constant for all simobjects, and a single static copy could be shared across multiple instances.

Class Members

double	m_dEquatorialRadius	Feet
double	m_dPolarRadius	Feet
float	m_fGravitySeaLevel	ft/s^2
float	m_fSpecificGasConstant	R
float	m_fSpecificHeatRatio	Gamma (Cp/Cv for air (specific heat ratio))
float	m_fStandardSeaLevelDensity	slugs/ft^3
float	m_fStandardSeaLevelPressure	Lbs/SqFt
float	m_fStandardSeaLevelTemperature	Rankine

◆ P3D::SurfaceInfoV400

class P3D::SurfaceInfoV400

Contains terrain/surface information for a given object's offset

Public Types

```
enum SURFACE_TYPE_CATEGORY {
    SURFACE_TYPE_CATEGORY_INVALID = -1,
    SURFACE_TYPE_CATEGORY_HARD,
    SURFACE_TYPE_CATEGORY_SOFT,
    SURFACE_TYPE_CATEGORY_WATER,
    SURFACE_TYPE_NUM_CATEGORIES
}

enum SURFACE_CONDITION_CATEGORY {
    SURFACE_CONDITION_CATEGORY_INVALID = -1,
    SURFACE_CONDITION_CATEGORY_NORMAL,
    SURFACE_CONDITION_CATEGORY_WET,
    SURFACE_CONDITION_CATEGORY_ICY,
    SURFACE_CONDITION_CATEGORY_SNOW,
    SURFACE_CONDITION_CATEGORY_LOOSE,
    SURFACE_CONDITION_CATEGORY_LOOSE_SNOW,
    SURFACE_CONDITION_NUM_CATEGORIES
}
```

Public Attributes

```
float m_fElevation
float m_fWaveHeight
FXYZ m_vNormal
FXYZ m_vVelocity
FXYZ m_vRotVelocity
BOOL m_bOnPlatform
SURFACE_TYPE_CATEGORY m_eSurfaceCategory
SURFACE_CONDITION_CATEGORY m_eSurfaceCondition
```

Member Enumeration Documentation

◆ SURFACE_CONDITION_CATEGORY

enum SURFACE_CONDITION_CATEGORY

Surface conditions

Enumerator

SURFACE_CONDITION_CATEGORY_INVALID

SURFACE_CONDITION_CATEGORY_NORMAL

SURFACE_CONDITION_CATEGORY_WET
SURFACE_CONDITION_CATEGORY_ICY
SURFACE_CONDITION_CATEGORY_SNOW
SURFACE_CONDITION_CATEGORY_LOOSE
SURFACE_CONDITION_CATEGORY_LOOSE_SNOW
SURFACE_CONDITION_NUM_CATEGORIES

◆ SURFACE_TYPE_CATEGORY

enum **SURFACE_TYPE_CATEGORY**

Surface categories

Enumerator

SURFACE_TYPE_CATEGORY_INVALID
SURFACE_TYPE_CATEGORY_HARD
SURFACE_TYPE_CATEGORY_SOFT
SURFACE_TYPE_CATEGORY_WATER
SURFACE_TYPE_NUM_CATEGORIES

Member Data Documentation

◆ m_bOnPlatform

BOOL m_bOnPlatform

TRUE or FALSE

◆ m_eSurfaceCategory

SURFACE_TYPE_CATEGORY m_eSurfaceCategory

SURFACE_TYPE_CATEGORY

◆ m_eSurfaceCondition

SURFACE_CONDITION_CATEGORY m_eSurfaceCondition

SURFACE_CONDITION_CATEGORY

◆ m_fElevation

float m_fElevation

Feet

◆ m_fWaveHeight

float m_fWaveHeight

Feet

◆ m_vNormal

FXYZ m_vNormal

Unit vector

◆ m_vRotVelocity

FXYZ m_vRotVelocity

Radians per Second

◆ m_vVelocity

FXYZ m_vVelocity

Feet per Second

◆ P3D::WeatherInfoV400

class P3D::WeatherInfoV400

Contains terrain/surface information for a given object's offset

Class Members

BOOL	m_bIsInCloud	TRUE or FALSE.
BOOL	m_bIsRaining	TRUE or FALSE.
BOOL	m_bIsSnowing	TRUE or FALSE.
BOOL	m_bUnsteadyWind	TRUE if gusting.
float	m_fAmbientPressure	PSF.
float	m_fCloudIntensityPercent	0.0 = none, 1.0 = max
float	m_fPrecipIntensityPercent	0.0 = min, 1.0 = max
float	m_fSeaLevelPressure	PSF.
float	m_fTemperature	Degrees Rankine.
float	m_fTurbulencePercent	0.0 = none, 1.0 = max
float	m_fVisibility	Feet.
float	m_fWindDirection	Radians True.
float	m_fWindSpeed	Feet per Second.
XYZ	m_vWind	Feet per Second.

◆ P3D::IMassPropertiesV01

class P3D::IMassPropertiesV01

Interface for getting mass properties from SimObject implementation

Inherits IMassProperties.

Private Member Functions

```
virtual float GetWeight() const PURE  
virtual BOOL RegisterMass(_in const IMassElement *pElement) PURE  
virtual BOOL UnRegisterMass(_in const IMassElement *pElement) PURE  
virtual void ForceUpdate() PURE
```

Member Function Documentation

◆ ForceUpdate()

virtual void ForceUpdate() [private] [virtual]

Force an update of properties

◆ GetWeight()

virtual float GetWeight() const [private] [virtual]

Returns weight of SimObject in pounds.

◆ RegisterMass()

virtual BOOL RegisterMass(_in const IMassElement * pElement) [private] [virtual]

Registers mass of SimObject.

◆ UnRegisterMass()

virtual BOOL UnRegisterMass(_in const IMassElement * pElement) [private] [virtual]

Unregisters mass of SimObject

◆ P3D::IForceMomentsV01

class P3D::IForceMomentsV01

Interface for getting physical forces from a SimObject implementation.

Inherits IForceMoments.

Private Member Functions

```
virtual BOOL RegisterElement (__in IForceElement *pElement) PURE  
virtual BOOL UnRegisterElement (__in IForceElement *pElement) PURE  
virtual UINT ElementCount () const PURE  
virtual IForceElement * GetElement (int index) const PURE
```

Member Function Documentation

◆ ElementCount()

```
virtual UINT ElementCount ( ) const [private] [virtual]
```

Returns the total number of force elements

◆ GetElement()

```
virtual IForceElement* GetElement ( int index ) const [private] [virtual]
```

Gets a force element

◆ RegisterElement()

```
virtual BOOL RegisterElement ( __in IForceElement * pElement ) [private] [virtual]
```

Registers a force element

◆ UnRegisterElement()

```
virtual BOOL UnRegisterElement ( __in IForceElement * pElement ) [private] [virtual]
```

Unregisters a force element

◆ P3D::ICollisionServiceV01

class P3D::ICollisionServiceV01

Interface for getting crash parameters for this object

Inherits ICollisionService.

Private Member Functions

```
virtual BOOL InvokesCrashOnOtherObjects () PURE
```

```
virtual void SetInvokesCrashOnOtherObjects ( __in BOOL invokesCrash ) PURE
```

Member Function Documentation

◆ InvokesCrashOnOtherObjects()

```
virtual BOOL InvokesCrashOnOtherObjects ( ) [private] [virtual]
```

Whether crash will invoke on other objects

◆ SetInvokesCrashOnOtherObjects()

```
virtual void SetInvokesCrashOnOtherObjects ( __in BOOL invokesCrash ) [private] [virtual]
```

Sets whether to invoke crash on other objects

◆ P3D::IAircraftServiceV01

class P3D::IAircraftServiceV01

Inherits IAircraftService.

Inherited by **IAirplaneServiceV01**, and **IRotorcraftServiceV01**.

Private Member Functions

```
virtual float GetIndicatedAirspeed () const PURE
```

Member Function Documentation

◆ **GetIndicatedAirspeed()**

virtual float GetIndicatedAirspeed() const **private** **virtual**

Gets the indicated airspeed. (feet per second)

◆ **P3D::IAirplaneServiceV01**

class P3D::IAirplaneServiceV01

Inherits **IAircraftServiceV01**.

◆ **P3D::IRotorcraftServiceV01**

class P3D::IRotorcraftServiceV01

Inherits **IAircraftServiceV01**.

◆ **P3D::IBoatServiceV01**

class P3D::IBoatServiceV01

Inherits **IBoatService**.

◆ **P3D::IGroundVehicleServiceV01**

class P3D::IGroundVehicleServiceV01

Inherits **IGroundVehicleService**.

◆ **P3D::IAtcServiceV01**

class P3D::IAtcServiceV01

Interface for getting ATC parameters from this object

Inherits **IAtcService**.

◆ **P3D::IRadarSignatureServiceV01**

class P3D::IRadarSignatureServiceV01

Interface for getting the radar signature of this object

Inherits **IRadarSignatureService**.

◆ **P3D::IDoorServiceV01**

class P3D::IDoorServiceV01

Interface for getting door parameters for this object

Inherits **IDoorService**.

Private Member Functions

virtual float **GetDoorPercentOpen** (__in int **doorIndex**) const **PURE**

Member Function Documentation

◆ **GetDoorPercentOpen()**

virtual float GetDoorPercentOpen (__in int **doorIndex**) const **private** **virtual**

Returns percentage of how open a door currently is

Parameters

doorIndex Index that associated with a door

◆ **P3D::IFuelServiceV400**

class P3D::IFuelServiceV400

Interface for setting/getting fuel levels, such as in the UI

Inherits IUnknown.

Private Member Functions

```
virtual float GetWeightPerGallon() const PURE
virtual UINT GetNumberOfTanks() const PURE
virtual float GetTotalCapacityGallons() const PURE
virtual float GetTotalLevelPercent() const PURE
virtual HRESULT SetTotalLevelPercent(float fPct) PURE
virtual HRESULT GetTankName(__in int iTankIndex, __out LPWSTR pszName, __in UINT uLength) const PURE
virtual float GetTankCapacityGallons(__in int iTankIndex) const PURE
virtual float GetTankLevelPercent(__in int iTankIndex) const PURE
virtual HRESULT SetTankLevelPercent(__in int iTankIndex, float fPct) PURE
```

Member Function Documentation

◆ GetNumberOfTanks()

virtual UINT GetNumberOfTanks() const [private] [virtual]

Number of tanks with capacity greater than zero.

◆ GetTankCapacityGallons()

virtual float GetTankCapacityGallons(__in int iTankIndex) const [private] [virtual]

Gallons for specified tank.

◆ GetTankLevelPercent()

virtual float GetTankLevelPercent(__in int iTankIndex) const [private] [virtual]

Percentage (0 - 1) for specified tank.

◆ GetTankName()

```
virtual HRESULT GetTankName(__in int iTankIndex,
                           __out LPWSTR pszName,
                           __in UINT uLength
                           ) const [private] [virtual]
```

Returns string name used in the Fuel User Interface for specified tank.

◆ GetTotalCapacityGallons()

virtual float GetTotalCapacityGallons() const [private] [virtual]

Total capacity for all tanks

◆ GetTotalLevelPercent()

virtual float GetTotalLevelPercent() const [private] [virtual]

Total percentage (0-1) for all tanks combined.

◆ GetWeightPerGallon()

virtual float GetWeightPerGallon() const [private] [virtual]

Pounds per gallon

◆ SetTankLevelPercent()

```
virtual HRESULT SetTankLevelPercent(__in int iTankIndex,
                                    float fPct
```

) **private** **virtual**

Sets the percentage (0 - 1) for specified tank.

◆ **SetTotalLevelPercent()**

virtual **HRESULT** **SetTotalLevelPercent** (**float** **fPct**) **private** **virtual**

Sets total percentage (0-1) for all tanks combined.

◆ **P3D::ISurfaceQueryManagerV400**

class P3D::ISurfaceQueryManagerV400

General surface queries, not associated with a specific object. This service interface is available from the PDK.

Inherits **IUnknown**.

Private Member Functions

virtual **HRESULT** **QuerySurfaceInformation** (**_out** **SurfaceInfoV400** & **SurfaceInfo**, **_in** **const** **DXYZ** & **vWorldPosRadiansFeet**) **const** **PURE**

virtual **HRESULT** **QuerySurfaceElevation** (**_out** **float** & **fElevationFeet**, **_in** **const** **DXYZ** & **vWorldPosRadiansFeet**) **const** **PURE**

virtual **HRESULT** **QueryBathymetryElevation** (**_out** **float** & **fDepthFeet**, **_in** **const** **DXYZ** & **vWorldPosRadiansFeet**) **const** **PURE**

Member Function Documentation

◆ **QueryBathymetryElevation()**

virtual **HRESULT**
QueryBathymetryElevation (**_out** **float** & **fDepthFeet**,
_in **const** **DXYZ** & **vWorldPosRadiansFeet**
) **const** **private** **virtual**

Provides bathymetry elevation (in feet) for a given world-relative position. (X = Longitude in radians, Y = Altitude in feet, Z = Latitude in radians). Returns **E_FAIL** if query fails.

◆ **QuerySurfaceElevation()**

virtual **HRESULT**
QuerySurfaceElevation (**_out** **float** & **fElevationFeet**,
_in **const** **DXYZ** & **vWorldPosRadiansFeet**
) **const** **private** **virtual**

Provides surface elevation (in feet) for a given world-relative position. (X = Longitude in radians, Y = Altitude in feet, Z = Latitude in radians). Returns **E_FAIL** if query fails.

◆ **QuerySurfaceInformation()**

virtual **HRESULT**
QuerySurfaceInformation (**_out** **SurfaceInfoV400** & **SurfaceInfo**,
_in **const** **DXYZ** & **vWorldPosRadiansFeet**
) **const** **private** **virtual**

Provides surface information for a given world-relative position. (X = Longitude in radians, Y = Altitude in feet, Z = Latitude in radians). Returns **E_FAIL** if query fails.

◆ **P3D::IWaypointQueryManagerV400**

class P3D::IWaypointQueryManagerV400

Interface that provides waypoint information specifically for the user in a mission. This service interface is available from the PDK.

Inherits **IUnknown**.

Private Member Functions

virtual **UINT** **GetNumberOfWaypointLists** () **const** **PURE**

virtual **HRESULT** **GetWaypointListIndexFromDescription** (**_in** **LPCWSTR** **pszDescription**,
_out **UINT** & **iWaypointList**) **const** **PURE**

virtual **UINT** **GetNumberOfWaypoints** (**_in** **UINT** **iWaypointList**) **const** **PURE**

```
VIRTUAL uint GetNumberOfWaypoints( ) const [private] [virtual]
virtual HRESULT GetWaypointListDescription( _in UINT iWaypointList, _out LPWSTR
pszDesc, _in UINT uLength) const PURE
virtual int GetWaypointID( _in UINT iWaypointList, _in UINT iWaypoint) const
PURE
virtual HRESULT GetWaypointDescription( _in UINT iWaypointList, _in UINT iWaypoint,
_out LPWSTR pszDescription, _in UINT uLength) const PURE
virtual HRESULT GetWaypointPosition( _in UINT iWaypointList, _in UINT iWaypoint, _out
P3D::DXYZ &vWorldPosition) const PURE
virtual HRESULT GetWaypointOrientation( _in UINT iWaypointList, _in UINT iWaypoint,
_out P3D::DXYZ &vOrientation) const PURE
virtual BOOL IsAltitudeAGL( _in UINT iWaypointList, _in UINT iWaypoint) const PURE
virtual HRESULT GetWaypointCustomValue( _in UINT iWaypointList, _in UINT iWaypoint,
_out LPWSTR pszCustomVal, _in UINT uLength) const PURE
```

Member Function Documentation

◆ GetNumberOfWaypointLists()

```
virtual uint GetNumberOfWaypointLists( ) const [private] [virtual]
```

Returns the number of waypoint lists the manager is holding.

◆ GetNumberOfWaypoints()

```
virtual uint GetNumberOfWaypoints( _in UINT iWaypointList) const [private] [virtual]
```

Returns the number of waypoints for the list specified by the list index input.

◆ GetWaypointCustomValue()

```
virtual HRESULT GetWaypointCustomValue( _in UINT iWaypointList,
_in UINT iWaypoint,
_out LPWSTR pszCustomVal,
_in UINT uLength
) const [private] [virtual]
```

Provides the text string specified in the Custom Value field for the waypoint specified by the list and waypoint index inputs. Returns E_FAIL if waypoint not found.

◆ GetWaypointDescription()

```
virtual HRESULT GetWaypointDescription( _in UINT iWaypointList,
_in UINT iWaypoint,
_out LPWSTR pszDescription,
_in UINT uLength
) const [private] [virtual]
```

Returns the specified description of the waypoint.

◆ GetWaypointID()

```
virtual int GetWaypointID( _in UINT iWaypointList,
_in UINT iWaypoint
) const [private] [virtual]
```

Returns the specified integer ID of the waypoint.

◆ GetWaypointListDescription()

```
virtual HRESULT GetWaypointListDescription( _in UINT iWaypointList,
_out LPWSTR pszDesc,
_in UINT uLength
) const [private] [virtual]
```

Provides the description string for the list specified by the list index input. Returns E_FAIL if waypoint list not found.

◆ GetWaypointListIndexFromDescription()

```
virtual HRESULT GetWaypointListIndexFromDescription(
_in LPCWSTR pszDescription,
_out uint &iWaypointList
```

) const private virtual

Provides the index (0 - based) of the waypoint list with the description string passed in. Returns E_FAIL if waypoint list not found.

◆ GetWaypointOrientation()

```
virtual HRESULT GetWaypointOrientation( __in UINT iWaypointList,
                                         __in UINT iWaypoint,
                                         __out P3D::DXYZ & vOrientation
                                         ) const private virtual
```

Provides the orientation for the waypoint specified by the list and waypoint index inputs. (X = Pitch in radians, Y = Heading in radians, Z = Bank in radians). Returns E_FAIL if waypoint not found.

◆ GetWaypointPosition()

```
virtual HRESULT GetWaypointPosition( __in UINT iWaypointList,
                                       __in UINT iWaypoint,
                                       __out P3D::DXYZ & vWorldPosition
                                       ) const private virtual
```

Provides the world-relative position for the waypoint specified by the list and waypoint index inputs. (X = Longitude in radians, Y = Altitude in feet, Z = Latitude in radians). Returns E_FAIL if waypoint not found.

◆ IsAltitudeAGL()

```
virtual BOOL IsAltitudeAGL( __in UINT iWaypointList,
                           __in UINT iWaypoint
                           ) const private virtual
```

Returns whether the specified altitude of the waypoint is Above Ground Level or Mean Sea Level.

◆ P3D::IAvatarSimV01

class P3D::IAvatarSimV01

This interface is to be implemented on any simulation object that is used for the user's avatar.

Inherits IUnknown.

Private Member Functions

```
virtual HRESULT OnAttach() PURE
virtual HRESULT OnDetach() PURE
```

Member Function Documentation

◆ OnAttach()

```
virtual HRESULT OnAttach() private virtual
```

Called when the avatar attaches to the user's object being controlled.

◆ OnDetach()

```
virtual HRESULT OnDetach() private virtual
```

Called when the avatar detaches to the user's object being controlled.

◆ P3D::IAnimationControllerV01

class P3D::IAnimationControllerV01

This service can be queried for on a simulation object to play pre-defined animations built into the 3-D visual model. For example, a walking animation.

Inherits IUnknown.

Private Member Functions

```
virtual HRESULT Play (const GUID &guidAnimationID, BOOL bLoop) PURE  
virtual HRESULT TransitionAndPlay (const GUID &guidCurrentAnimationID, const GUID  
&guidNextAnimationID, BOOL bLoop, double fBlendDuration) PURE
```

Member Function Documentation

◆ Play()

```
virtual HRESULT Play ( const GUID & guidAnimationID,  
                      BOOL          bLoop  
                    )
```

private virtual

Called to invoke a specified animation.

◆ TransitionAndPlay()

```
virtual HRESULT TransitionAndPlay ( const GUID & guidCurrentAnimationID,  
                                    const GUID & guidNextAnimationID,  
                                    BOOL          bLoop,  
                                    double        fBlendDuration  
                                  )
```

private virtual

Called to transition from one animation to another.

◆ P3D::IAvatarAttachServiceV01

class P3D::IAvatarAttachServiceV01

This service allows configuring conditions or constraints in which the avatar can be attached and detached on this object. For example, implement this on an airplane to prevent detaching at high speeds.

Inherits IUnknown.

Private Member Functions

```
virtual BOOL CanAvatarAttach () const PURE  
virtual BOOL CanAvatarDetach () const PURE
```

Member Function Documentation

◆ CanAvatarAttach()

```
virtual BOOL CanAvatarAttach ( ) const private virtual
```

Return if conditions are appropriate for attaching.

◆ CanAvatarDetach()

```
virtual BOOL CanAvatarDetach ( ) const private virtual
```

Return if conditions are appropriate for detaching.

◆ P3D::IMarkerManagerV310

class P3D::IMarkerManagerV310

This service allows for placing a graphical orthogonal marker on a simobject at a specified offset. This is useful for visualizing physical offsets relative to the visual model. For example, wheel and engine positions. By default, the marker consists of red 30 meter orthogonal lines in both the positive and negative directions of the X, Y, and Z axis. This service is available through the PDK service provider.

Inherits IMarkerManagerV01.

Private Member Functions

```
virtual HRESULT CreateObjectMarker (__in UINT idObject, __out HANDLE &hHandle)  
PURE  
virtual HRESULT UpdateObjectMarkerOffset (__in const HANDLE hHandle, __in const FXYZ  
&vOffset) PURE  
virtual HRESULT UpdateObjectMarkerOrientation (__in const HANDLE hHandle, __in const  
FXYZ &vOrientation) PURE
```

```
virtual HRESULT UpdateObjectMarkerOffsetAndOrientation(_in const HANDLE hHandle,
                                                     _in const FXYZ &vOffset, _in const FXYZ &vOrientation) PURE  
virtual HRESULT RemoveMarker(_inout HANDLE &hHandle) PURE
```

Member Function Documentation

◆ CreateObjectMarker()

```
virtual HRESULT CreateObjectMarker (_in UINT idObject,
                                    _out HANDLE & hHandle  
)
```

private virtual

Creates a new marker with the manager. It is advised to pass "0" for the handle, as that will verify with the manager that this is a new marker. A valid handle will be returned when successfully created. idObject is the Object ID in which to attach the marker.

◆ RemoveMarker()

```
virtual HRESULT RemoveMarker (_inout HANDLE & hHandle) private virtual
```

Called to remove the marker. This will unregister the marker. The handle will be returned to a value of "0", and use of the original should be avoided.

◆ UpdateObjectMarkerOffset()

```
virtual HRESULT UpdateObjectMarkerOffset (_in const HANDLE hHandle,
                                         _in const FXYZ & vOffset  
)
```

private virtual

This is called to update the offset from the object's center in which to draw the marker.

◆ UpdateObjectMarkerOffsetAndOrientation()

```
virtual HRESULT UpdateObjectMarkerOffsetAndOrientation (_in const HANDLE hHandle,
                                                       _in const FXYZ & vOffset,
                                                       _in const FXYZ & vOrientation  
)
```

private virtual

This is called to update both the offset and orientation from the object's center and body axis in which to draw the marker.

◆ UpdateObjectMarkerOrientation()

```
virtual HRESULT UpdateObjectMarkerOrientation (_in const HANDLE hHandle,
                                              _in const FXYZ & vOrientation  
)
```

private virtual

This is called to update the orientation relative to the object's body axis in which to draw the marker.

◆ P3D::IDesignatorServiceV340

class P3D::IDesignatorServiceV340

Interface implemented on a SimObject in order for core Prepar3D to interface with it for the purposes of broadcasting DIS related PDUs. This interface may also be queried by an ISimObject to gather designator related information.

Inherits IUnknown.

Private Member Functions

```
virtual UINT GetDesignatorCount () const PURE  
virtual BOOL IsActive (_in UINT iDesignator) const PURE  
virtual USHORT GetCodeName (_in UINT iDesignator) const PURE  
virtual UINT GetDesignatedObjectId (_in UINT iDesignator) const PURE  
virtual USHORT GetDesignatorCode (_in UINT iDesignator) const PURE  
virtual float GetDesignatorPower (_in UINT iDesignator) const PURE  
virtual float GetDesignatorWaveLength (_in UINT iDesignator) const PURE  
virtual HRESULT GetDesignatorSpotLocation (_in UINT iDesignator, _out DXYZ  
&vWorldPosRadiansFeet) const PURE  
virtual HRESULT GetDesignatorSpotAcceleration (_in UINT iDesignator, _out DXYZ
```

&vWorldAccelerationFps) const **PURE**

Member Function Documentation

◆ GetCodeName()

virtual USHORT GetCodeName (__in UINT iDesignator) const private virtual

The DIS code name

◆ GetDesignatedObjectId()

virtual UINT GetDesignatedObjectId (__in UINT iDesignator) const **private** **virtual**

The object ID of the designated target

- ◆ GetDesignatorCode()

virtual USHORT GetDesignatorCode (__in UINT iDesignator) const **private** **virtual**

The DIS designator code

- ◆ GetDesignatorCount()

virtual **UINT** GetDesignatorCount() const **private** **virtual**

The number of designators

- ◆ GetDesignatorPower()

virtual float GetDesignatorPower (__in UINT iDesignator) const **private** **virtual**

The power of the designator in watts

◆ GetDesignatorSpotAcceleration()

The world acceleration of the spot designation in feet/sec

◆ GetDesignatorSpotLocation()

virtual HRESULT
GetDesignatorSpotLocation

) const private virtual

The world location of the spot designation in radians/feet

◆ GetDesignatorWaveLength()

```
virtual float GetDesignatorWaveLength ( __in UINT iDesignator ) const private virtual
```

The wavelength of the designator in microns

◆ IsActive()

virtual **BOOL** IsActive (__in **UINT** iDesignator) const **private** **virtual**

The active state of the given designator

◆ P3D::IRayTraceManagerV340

class P3D::IRayTraceManagerV340

This service allows the user to perform collision based ray tracing. Ray tracing can be performed on either objects, terrain, or both based on the given interrogation type. When casting from an object

location, that object's object id should be in the ignore field to prevent it from casting on itself.

Object interrogation is typically more expensive than terrain interrogation. Ray length and granularity can be used to help control performance depending on the need of the ray trace. Ray trace calls should typically be done on both objects and terrain at a shorted ray length and a more precise granularity first. If no collision is detected, a higher ray length and less precise terrain based ray trace should be performed.

Note

Ray tracing is an expensive operation. Ray trace calls should be limited whenever possible.

Inherits IUnknown.

Private Member Functions

```
virtual HRESULT InterrogateWorldRay (_in DWORD dwInterrogationTypes, _in_opt UINT
    ignoreObjectld, _in const DXYZ &vWorldRadiansFeet, _in const DXYZ
    &xyzWorldUnitRayDir, _in float fRayLengthMax, _in float fGranularityMin,
    _out_opt UINT *pResultObjectld, _out_opt DXYZ
    *pResultWorldRadiansFeet, _inout DWORD &dwInterrogationResults) const
    PURE
```

Member Function Documentation

◆ InterrogateWorldRay()

```
virtual HRESULT
InterrogateWorldRay (_in DWORD dwInterrogationTypes,
    _in_opt UINT ignoreObjectld,
    _in const DXYZ & vWorldRadiansFeet,
    _in const DXYZ & xyzWorldUnitRayDir,
    _in float fRayLengthMax,
    _in float fGranularityMin,
    _out_opt UINT * pResultObjectld,
    _out_opt DXYZ * pResultWorldRadiansFeet,
    _inout DWORD & dwInterrogationResults
)
    const
private virtual
```

Performs a world space based collision ray trace.

Parameters

dwInterrogationTypes	INTERROGATIONTYPE flags. See Types.h
ignoreObjectld	This object will be ignored when performing the ray trace, likely the casting object (Optional)
vWorldRadiansFeet	The initial LonAltLat of the ray trace in world radians/feet
xyzWorldUnitRayDir	A unit vector representing the orientation in world space
fRayLengthMax	The maximum length of the ray trace in meters
fGranularityMin	The minimum step distance of the ray trace in meters
pResultObjectld	The resulting object id of the collision (Optional)
pResultWorldRadiansFeet	The resulting LonAltLat of the collision in world radians/feet (Optional)
dwInterrogationResults	The resulting INTERROGATIONTYPE flags. See Types.h

◆ P3D::IEmissionsServiceV340

class P3D::IEmissionsServiceV340

Interface implemented on a SimObject in order for core Prepar3D to interface with it for the purposes of broadcasting DIS related PDUs. This interface may also be queried by an ISimObject to gather electromagnetic emission related information.

Inherits IUnknown.

Private Member Functions

```
virtual UINT GetEmissionSystemCount () const PURE
virtual UINT GetEmitterBeamCount (_in UINT iSystem) const PURE
virtual UINT GetEmissionSystemName (_in UINT iSystem) const PURE
virtual UINT GetEmissionSystemFunction (_in UINT iSystem) const PURE
virtual UINT GetEmitterNumber (_in UINT iSystem) const PURE
virtual HRESULT GetEmissionSystemBodyOffset (_in UINT iSystem, _out
    P3D::P3DDXYZ &vBodyOffsetFeet) const PURE
virtual UINT GetTrackJamCount (_in UINT iSystem, _in UINT iBeam) const PURE
virtual UINT GetEmitterBeamNumber (_in UINT iSystem, _in UINT iBeam) const
    PURE
virtual UINT GetEmitterBeamParameter (_in UINT iSystem, _in UINT iBeam) const
    PURE
virtual float GetEmitterBeamFrequency (_in UINT iSystem, _in UINT iBeam) const
    PURE
virtual float GetEmitterBeamFrequencyRange (_in UINT iSystem, _in UINT iBeam)
    const PURE
```

```
virtual float GetEmitterBeamEffectiveRadiatedPower(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual float GetEmitterBeamPulseRepetitionFrequency(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual float GetEmitterBeamPulseWidth(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual float GetEmitterBeamAzimuthCenter(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual float GetEmitterBeamAzimuthSweep(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual float GetEmitterBeamElevationCenter(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual float GetEmitterBeamElevationSweep(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual float GetEmitterBeamSweepSync(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual uint GetEmitterBeamFunction(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual bool GetEmitterBeamIsHighDensityTrack(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual uint GetEmitterBeamJammingMode(__in UINT iSystem, __in UINT iBeam) const PURE  
virtual uint GetTrackJamObjectid(__in UINT iSystem, __in UINT iBeam, __in UINT iTrackJam) const PURE  
virtual uint GetTrackJamEmitterNumber(__in UINT iSystem, __in UINT iBeam, __in UINT iTrackJam) const PURE  
virtual uint GetTrackJamBeamNumber(__in UINT iSystem, __in UINT iBeam, __in UINT iTrackJam) const PURE
```

Member Function Documentation

◆ GetEmissionSystemBodyOffset()

```
virtual HRESULT  
GetEmissionSystemBodyOffset( _in UINT iSystem,  
                           _out P3DDXYZ & vBodyOffsetFeet  
                           ) const {  
    return E_NOTIMPL;  
}
```

The offset of the beam source in body coordinates (feet)

◆ GetEmissionSystemCount()

virtual UINT GetEmissionSystemCount () const

The number of emission systems

◆ GetEmissionSystemFunction()

virtual UINT GetEmissionSystemFunction (__in UINT iSystem) const **private** **virtual**

The DIS emission system function

◆ GetEmissionSystemName()

virtual UINT GetEmissionSystemName (__in UINT iSystem) const **private** **virtual**

The DIS emission system name

◆ GetEmitterBeamAzimuthCenter()

```
virtual float GetEmitterBeamAzimuthCenter ( __in UINT iSystem,  
                                            __in UINT iBeam  
                                         ) const
```

The center azimuth of the beam in radians relative to the emitter

◆ GetEmitterBeamAzimuthSweep()

The half-angle sweep of the azimuth in radians relative to the center azimuth

◆ **GetEmitterBeamCount()**

```
virtual UINT GetEmitterBeamCount ( __in UINT iSystem ) const [private] [virtual]
```

The number of emitter beam in the given emission system

◆ **GetEmitterBeamEffectiveRadiatedPower()**

```
virtual float GetEmitterBeamEffectiveRadiatedPower ( __in UINT iSystem,
                                                    __in UINT iBeam
                                                    ) const [private] [virtual]
```

The average effective radiated power of the beam in dBm

◆ **GetEmitterBeamElevationCenter()**

```
virtual float GetEmitterBeamElevationCenter ( __in UINT iSystem,
                                              __in UINT iBeam
                                              ) const [private] [virtual]
```

The center elevation of the beam in radians relative to the emitter

◆ **GetEmitterBeamElevationSweep()**

```
virtual float GetEmitterBeamElevationSweep ( __in UINT iSystem,
                                              __in UINT iBeam
                                              ) const [private] [virtual]
```

The half-angle sweep of the elevation in radians relative to the center elevation

◆ **GetEmitterBeamFrequency()**

```
virtual float GetEmitterBeamFrequency ( __in UINT iSystem,
                                         __in UINT iBeam
                                         ) const [private] [virtual]
```

The frequency of the beam in hertz

◆ **GetEmitterBeamFrequencyRange()**

```
virtual float GetEmitterBeamFrequencyRange ( __in UINT iSystem,
                                              __in UINT iBeam
                                              ) const [private] [virtual]
```

The frequency range of the beam in hertz

◆ **GetEmitterBeamFunction()**

```
virtual UINT GetEmitterBeamFunction ( __in UINT iSystem,
                                       __in UINT iBeam
                                       ) const [private] [virtual]
```

The DIS beam function

◆ **GetEmitterBeamIsHighDensityTrack()**

```
virtual BOOL GetEmitterBeamIsHighDensityTrack ( __in UINT iSystem,
                                                __in UINT iBeam
                                                ) const [private] [virtual]
```

True if all targets in the scan pattern are to be considered tracked or jammed

◆ **GetEmitterBeamJammingMode()**

```
virtual UINT GetEmitterBeamJammingMode ( __in UINT iSystem,
                                         __in UINT iBeam
                                         ) const [private] [virtual]
```

The DIS jamming mode sequence for the given emitter in string representation

◆ GetEmitterBeamNumber()

The beam identification number

◆ GetEmitterBeamParameter()

The DIS beam parameter

◆ GetEmitterBeamPulseRepetitionFrequency()

```
virtual float GetEmitterBeamPulseRepetitionFrequency ( __in UINT iSystem,
                                                       __in UINT iBeam
) const private virtual
```

The average pulse repetition frequency of the beam in hertz

◆ GetEmitterBeamPulseWidth()

The average pulse width of the beam in microseconds

◆ GetEmitterBeamSweepSync()

The range from 0.0 to 100.0 representing the percentage of the pattern scanned

◆ GetEmitterNumber()

virtual UINT GetEmitterNumber (in UINT iSystem) const private virtual

The emitter identification number

◆ GetTrackJamBeamNumber()

The beam identification number associated with the target

◆ GetTrackJamCount()

The number targets being tracked/jammed

◆ GetTrackIamEmitterNumber()

virtual UINT GetTrackJamEmitterNumb

```
    __in UINT iBeam,  
    __in UINT iTrackJam  
) const [private] vi
```

The emitter identification number associated with the target

◆ GetTrackJamObjectId()

```
virtual UINT GetTrackJamObjectId( __in UINT iSystem,  
                                 __in UINT iBeam,  
                                 __in UINT iTrackJam  
                               ) const [private] [virtual]
```

The object id of the target being tracked/jammed

◆ P3D::IRadioSystemV400

```
class P3D::IRadioSystemV400
```

IRadioSystemV400 This interface can be queried for on any object to determine if a radio system is available.

Inherits IUnknown.

Private Member Functions

```
virtual BOOL AreRadiosActive() const [PURE]
```

Member Function Documentation

◆ AreRadiosActive()

```
virtual BOOL AreRadiosActive() const [private] [virtual]
```

Queries if any radios are currently active

Returns

TRUE if any radios active

◆ P3D::IAttachmentServiceV430

```
class P3D::IAttachmentServiceV430
```

IAttachmentServiceV430 This interface can be queried for on an object to obtain attachment data.

Inherits IAttachmentServiceV420.

Private Member Functions

```
virtual HRESULT GetAttachPointIndex( __in __notnull LPCSTR pszAttachPointName, __out  
                                    UINT &ulIndex) const [PURE]
```

```
virtual HRESULT GetAttachPointCount( __out UINT &uCount) const [PURE]
```

```
virtual HRESULT GetAttachPointOffset( __in UINT ulIndex, __out DXYZ &vOffsetMeters)  
                                    const [PURE]
```

```
virtual HRESULT GetAttachPointOrientation( __in UINT ulIndex, __out DXYZ  
                                         &vOrientationRadians) const [PURE]
```

```
virtual HRESULT GetAttachedObjectCount( __out UINT &nObjects) const [PURE]
```

```
virtual HRESULT GetAttachedObjects( __inout UINT &nObjects, __out UINT *rgObjectIDs)  
                                    const [PURE]
```

```
virtual HRESULT GetAttachedObjectIndex( __in UINT uObjectId, __out UINT &ulIndex) const  
                                         [PURE]
```

```
virtual HRESULT GetAttachedObjectId( __in UINT ulIndex, __out UINT &uObjectId) const  
                                         [PURE]
```

```
virtual HRESULT UpdateAttachments() [PURE]
```

```
    virtual UINT GetParentId() [PURE]
```

```
virtual HRESULT SetOffsetFeet( __in const DXYZ &vOffsetFeet) [PURE]
```

```
virtual HRESULT GetOffsetFeet( __out DXYZ &vOffsetFeet) [PURE]
```

```
virtual HRESULT SetOffsetRadians( __in const DXYZ &vOffsetRadians) [PURE]
```

```
virtual HRESULT GetOffsetRadians( __out DXYZ &vOffsetRadians) [PURE]
```

Member Function Documentation

◆ GetAttachedObjectCount()

```
virtual HRESULT GetAttachedObjectCount( __out UINT & nObjects) const [private] [virtual]
```

Gets the number of attached objects.

Parameters

nObjects The number of attached objects.

Returns

S_OK if the attached objects were successfully found, E_FAIL otherwise.

◆ GetAttachedObjectId()

```
virtual HRESULT GetAttachedObjectId ( __in UINT      ulIndex,
                                     __out UINT & uObjectId
                                     )           const    [private] [virtual]
```

Gets the attached object id for the given attach point index.

Parameters

ulIndex The attach point index.

uObjectId The attached object's id.

Returns

S_OK if the attach point index's attached object id is successfully found, E_FAIL otherwise.

◆ GetAttachedObjectIndex()

```
virtual HRESULT GetAttachedObjectIndex ( __in UINT      uObjectId,
                                         __out UINT & ulIndex
                                         )           const    [private] [virtual]
```

Gets the attach point index for the given attached object id.

Parameters

uObjectId The attached object's id.

ulIndex The attached object's attach point index.

Returns

S_OK if the attached object's attach point index is successfully found, E_FAIL otherwise.

◆ GetAttachedObjects()

```
virtual HRESULT GetAttachedObjects ( __inout UINT & nObjects,
                                    __out UINT * rgObjectIDs
                                    )           const    [private] [virtual]
```

Gets a list of all attached objects.

Parameters

nObjects IN: The max number of objects requested. This must be no smaller than the size of the array pointed to by rgObjectIDs.

nObjects OUT: The actual number of objects found.

rgObjectIDs Address of array in which object IDs are returned.

Returns

S_OK if the attached objects were successfully found, E_FAIL otherwise.

Note

It is the callers responsibility to allocate the array's required memory.

◆ GetAttachPointCount()

```
virtual HRESULT GetAttachPointCount ( __out UINT & uCount ) const [private] [virtual]
```

Gets the attach point count.

Parameters

uCount The number of attach points.

Returns

S_OK if the attach point count was successfully found, E_FAIL otherwise.

◆ GetAttachPointIndex()

```
virtual HRESULT GetAttachPointIndex ( __in __notnull LPCSTR pszAttachPointName,
                                      __out UINT & ulIndex
                                      )           const    [private] [virtual]
```

Gets the attach point index from the given name.

Parameters

pszAttachPointName The name of the attach point.

ulIndex The index of the given attach point.

Returns

S_OK if the attach point index was successfully found, E_FAIL otherwise.

◆ GetAttachPointOffset()

```
virtual HRESULT GetAttachPointOffset ( __in UINT ulIndex,
                                     __out DXYZ & vOffsetMeters
                                     ) const [private] [virtual]
```

Gets the offset of the attach point with the given index.

Parameters

ulIndex The attach point index.
vOffsetMeters The body offset of the attach point in meters.

Returns

S_OK if the attach point offset was successfully found, E_FAIL otherwise.

Note

Units for this function are in meters, not feet.

◆ GetAttachPointOrientation()

```
virtual HRESULT GetAttachPointOrientation ( __in UINT ulIndex,
                                           __out DXYZ & vOrientationRadians
                                           ) const [private] [virtual]
```

Gets the orientation of the attach point with the given index.

Parameters

ulIndex The attach point index.
vOrientationRadians The body orientation offset of the attach point in radians.

Returns

S_OK if the attach point orientation was successfully found, E_FAIL otherwise.

◆ GetOffsetFeet()

```
virtual HRESULT GetOffsetFeet ( __out DXYZ & vOffsetFeet ) [private] [virtual]
```

If this object is attached to another object, this function will provide the offset in feet relative to the parent object and return S_OK. If this object is not attached to another object, this function will return E_FAIL.

Parameters

vOffsetFeet The offset relative to the parent object in feet.

Returns

S_OK if attached and the offset is valid, E_FAIL otherwise.

◆ GetOffsetRadians()

```
virtual HRESULT GetOffsetRadians ( __out DXYZ & vOffsetRadians ) [private] [virtual]
```

If this object is attached to another object, this function will provide the orientation offset in radians relative to the parent object and return S_OK. If this object is not attached to another object, this function will return E_FAIL.

Parameters

vOffsetRadians The orientation offset relative to the parent object in radians.

Returns

S_OK if attached and the offset is valid, E_FAIL otherwise.

◆ GetParentId()

```
virtual UINT GetParentId ( ) [private] [virtual]
```

If this object is attached to another object, this function will return the parent's object ID. If this object is not attached to another object, this function will return zero.

Returns

The parent's object id if attached, zero otherwise.

◆ SetOffsetFeet()

```
virtual HRESULT SetOffsetFeet ( __in const DXYZ & vOffsetFeet ) [private] [virtual]
```

If this object is attached to another object, this function will set the offset in feet relative to the

◆ IsBehaviorActive()

virtual **BOOL** IsBehaviorActive (__in const GUID & BehaviorGuid) const **private** **virtual**

Returns whether a behavior is active (true) or inactive (false)

◆ P3D::IAIBehaviorWingmanFormationV01

class P3D::IAIBehaviorWingmanFormationV01

Professional Plus Only

Interface for the Wingman Formation AI Behavior

Inherits IAIBehaviorWingmanFormation.

Private Member Functions

virtual GUID **GetBehaviorGuid** () const **PURE**

virtual void **SetLeaderObjectID** (int objectID) **PURE**

virtual int **GetLeaderObjectID** () const **PURE**

virtual void **SetOffsetPosition** (const P3D::DXYZ &offsetFeet) **PURE**

virtual void **GetOffsetPosition** (P3D::DXYZ &offsetFeet) const **PURE**

virtual void **WarpToTarget** () **PURE**

Member Function Documentation

◆ GetBehaviorGuid()

virtual GUID GetBehaviorGuid () const **private** **virtual**

Gets the Guid ID of a behavior

◆ GetLeaderObjectID()

virtual int GetLeaderObjectID () const **private** **virtual**

Gets the ID of the object's leader object

◆ GetOffsetPosition()

virtual void GetOffsetPosition (P3D::DXYZ & offsetFeet) const **private** **virtual**

Gets the offset position of a behavior (in feet)

◆ SetLeaderObjectID()

virtual void SetLeaderObjectID (int objectID) **private** **virtual**

Sets the ID of the object's leader object

◆ SetOffsetPosition()

virtual void SetOffsetPosition (const P3D::DXYZ & offsetFeet) **private** **virtual**

Sets the offset position of a behavior (in feet)

◆ WarpToTarget()

virtual void WarpToTarget () **private** **virtual**

Moves the object to its target's location

◆ P3D::IAIBehaviorAttackerV400

class P3D::IAIBehaviorAttackerV400

Professional Plus Only

Interface for the Attacker AI Behavior

Inherits IAIBehavior.

Private Member Functions

```
virtual GUID GetBehaviorGuid() const PURE
virtual void SetFireRadiusMin(float radiusFeet) PURE
virtual float GetFireRadiusMin() const PURE
virtual void SetFireRadiusMax(float radiusFeet) PURE
virtual float GetFireRadiusMax() const PURE
virtual void SetFireFOVDegrees(float degrees) PURE
virtual float GetFireFOVDegrees() const PURE
virtual void SetWeaponTitle(WCHAR *weaponTitle) PURE
virtual const WCHAR * GetWeaponTitle() const PURE
virtual void SetWeaponType(WCHAR *weaponType) PURE
virtual const WCHAR * GetWeaponType() const PURE
virtual void SetGunTitle(WCHAR *gunTitle) PURE
virtual const WCHAR * GetGunTitle() const PURE
virtual void SetGunType(WCHAR *gunType) PURE
virtual const WCHAR * GetGunType() const PURE
virtual void SetAttackDelay(float delaySeconds) PURE
virtual float GetAttackDelay() const PURE
virtual void SetGunBurstDuration(float durationSeconds) PURE
virtual float GetGunBurstDuration() const PURE
virtual BOOL IsWithinFireZone() const PURE
```

Member Function Documentation

◆ GetAttackDelay()

virtual float GetAttackDelay() const **private** **virtual**

Gets the delay between an attacker's attacks (in seconds)

◆ GetBehaviorGuid()

virtual GUID GetBehaviorGuid() const **private** **virtual**

Gets the Guid ID of a behavior

◆ GetFireFOVDegrees()

virtual float GetFireFOVDegrees() const **private** **virtual**

Gets the fire FOV of an attacker (in degrees)

◆ GetFireRadiusMax()

virtual float GetFireRadiusMax() const **private** **virtual**

Gets the maximum fire radius of an attacker (in feet)

◆ GetFireRadiusMin()

virtual float GetFireRadiusMin() const **private** **virtual**

Gets the minimum fire radius of an attacker (in feet)

◆ GetGunBurstDuration()

virtual float GetGunBurstDuration() const **private** **virtual**

Gets the duration of an attacker's gun burst (in seconds)

◆ GetGunTitle()

virtual const WCHAR* GetGunTitle() const **private** **virtual**

Gets the title of an attacker's gun

- ◆ **GetGunType()**
virtual const WCHAR* GetGunType() const **private** **virtual**
Gets the type of an attacker's gun
- ◆ **GetWeaponTitle()**
virtual const WCHAR* GetWeaponTitle() const **private** **virtual**
Gets the title of an attacker's weapon
- ◆ **GetWeaponType()**
virtual const WCHAR* GetWeaponType() const **private** **virtual**
Gets the type of an attacker's weapon
- ◆ **IsWithinFireZone()**
virtual **BOOL** IsWithinFireZone() const **private** **virtual**
Returns true if attacker is within fire zone, false otherwise
- ◆ **SetAttackDelay()**
virtual void SetAttackDelay(float **delaySeconds**) **private** **virtual**
Sets the delay between an attacker's attacks (in seconds)
- ◆ **SetFireFOVDegrees()**
virtual void SetFireFOVDegrees(float **degrees**) **private** **virtual**
Sets the fire FOV of an attacker (in degrees)
- ◆ **SetFireRadiusMax()**
virtual void SetFireRadiusMax(float **radiusFeet**) **private** **virtual**
Sets the maximum fire radius of an attacker (in feet)
- ◆ **SetFireRadiusMin()**
virtual void SetFireRadiusMin(float **radiusFeet**) **private** **virtual**
Sets the minimum fire radius of an attacker (in feet)
- ◆ **SetGunBurstDuration()**
virtual void SetGunBurstDuration(float **durationSeconds**) **private** **virtual**
Sets the duration of an attacker's gun burst (in seconds)
- ◆ **SetGunTitle()**
virtual void SetGunTitle(WCHAR * **gunTitle**) **private** **virtual**
Sets the title of an attacker's gun
- ◆ **SetGunType()**
virtual void SetGunType(WCHAR * **gunType**) **private** **virtual**
Sets the type of an attacker's gun
- ◆ **SetWeaponTitle()**
virtual void SetWeaponTitle(WCHAR * **weaponTitle**) **private** **virtual**

Sets the title of an attacker's weapon

◆ **SetWeaponType()**

virtual void SetWeaponType (WCHAR * weaponType) **private** **virtual**

Sets the type of an attacker's weapon

◆ **P3D::IAIBehaviorPursueV500**

class P3D::IAIBehaviorPursueV500

Professional Plus Only

Interface for the Pursue AI Behavior

Inherits IAIBehaviorPursueV01.

Private Member Functions

```
virtual GUID GetBehaviorGuid () const PURE
virtual void SetInvestigateRadius (float radiusFeet) PURE
virtual float GetInvestigateRadius () const PURE
virtual void SetStoppingDistance (float stoppingFeet) PURE
virtual float GetStoppingDistance () const PURE
virtual void SetPursueObjectID (int objectID) PURE
virtual int GetPursueObjectID () const PURE
virtual void SetMaxPursuitDurationSeconds (float maxSeconds) PURE
virtual float GetSetMaxPursuitDurationSeconds () const PURE
virtual void SetInterceptGroundObjects (BOOL intercept) PURE
virtual BOOL GetInterceptGroundObjects () const PURE
```

Member Function Documentation

◆ **GetBehaviorGuid()**

virtual GUID GetBehaviorGuid () const **private** **virtual**

Gets the Guid ID of a pursue behavior

◆ **GetInterceptGroundObjects()**

virtual BOOL GetInterceptGroundObjects () const **private** **virtual**

Gets if object can intercept ground objects

◆ **GetInvestigateRadius()**

virtual float GetInvestigateRadius () const **private** **virtual**

Gets the how far an object will investigate from its current location (in feet)

◆ **GetPursueObjectID()**

virtual int GetPursueObjectID () const **private** **virtual**

Gets the ID of the object being pursued

◆ **GetSetMaxPursuitDurationSeconds()**

virtual float GetSetMaxPursuitDurationSeconds () const **private** **virtual**

Gets the maximum time an object will pursue for (in seconds)

◆ **GetStoppingDistance()**

virtual float GetStoppingDistance () const **private** **virtual**

Gets the how far an object will stop from the AI object being pursued (in feet)

◆ **SetInterceptGroundObjects()**
virtual void SetInterceptGroundObjects (**BOOL** intercept) **private** **virtual**

Sets if object can intercept ground objects

◆ **SetInvestigateRadius()**
virtual void SetInvestigateRadius (float radiusFeet) **private** **virtual**

Sets the how far an object will investigate from its current location (in feet)

◆ **SetMaxPursuitDurationSeconds()**
virtual void SetMaxPursuitDurationSeconds (float maxSeconds) **private** **virtual**

Sets the maximum time an object will pursue for (in seconds)

◆ **SetPursueObjectID()**
virtual void SetPursueObjectID (int objectID) **private** **virtual**

Sets the ID of the object being pursued

◆ **SetStoppingDistance()**
virtual void SetStoppingDistance (float stoppingFeet) **private** **virtual**

Sets the how far an object will stop from the AI object being pursued (in feet)

◆ P3D::IAIBehaviorCombatAirPatrolV01

class P3D::IAIBehaviorCombatAirPatrolV01

Professional Plus Only

Interface for the Combat-Air-Patrol AI Behavior

Inherits IAIBehaviorCombatAirPatrol.

Private Member Functions

virtual GUID **GetBehaviorGuid** () const **PURE**
virtual void **SetPatrolObjectID** (int objectID) **PURE**
virtual int **GetPatrolObjectID** () const **PURE**
virtual void **SetPatrolOrigin** (const P3D::DXYZ &lonAltLat) **PURE**
virtual void **GetPatrolOrigin** (P3D::DXYZ &lonAltLat) const **PURE**
virtual void **SetPatrolRadius** (float radiusFeet) **PURE**
virtual float **GetPatrolRadius** () const **PURE**

Member Function Documentation

◆ **GetBehaviorGuid()**
virtual GUID GetBehaviorGuid () const **private** **virtual**

Gets the Guid ID of a combat-air-patrol behavior

◆ **GetPatrolObjectID()**
virtual int GetPatrolObjectID () const **private** **virtual**

Gets the ID of the patrol object

◆ **GetPatrolOrigin()**
virtual void GetPatrolOrigin (P3D::DXYZ & lonAltLat) const **private** **virtual**

Gets the origin point for the patrol (in radians and feet)

◆ **GetPatrolRadius()**

```
virtual float GetPatrolRadius( ) const private virtual
```

Gets the radius for the patrol (in feet)

◆ **SetPatrolObjectID()**

```
virtual void SetPatrolObjectID( int objectID ) private virtual
```

Sets the ID of the patrol object

◆ **SetPatrolOrigin()**

```
virtual void SetPatrolOrigin( const P3D::DXYZ & lonAltLat ) private virtual
```

Sets the origin point for the patrol (in radians and feet)

◆ **SetPatrolRadius()**

```
virtual void SetPatrolRadius( float radiusFeet ) private virtual
```

Sets the radius for the patrol (in feet)

◆ **P3D::IAIBehaviorCloseAirSupportV01**

```
class P3D::IAIBehaviorCloseAirSupportV01
```

Professional Plus Only

Interface for the Close-Air-Support AI Behavior

Inherits IAIBehaviorCloseAirSupport.

Private Member Functions

```
virtual GUID GetBehaviorGuid() const PURE  
virtual void SetSupportObjectID( int objectID ) PURE  
virtual int GetSupportObjectID() const PURE  
virtual void SetSupportPosition( const P3D::DXYZ &lonAltLat ) PURE  
virtual void GetSupportPosition( P3D::DXYZ &lonAltLat ) const PURE
```

Member Function Documentation

◆ **GetBehaviorGuid()**

```
virtual GUID GetBehaviorGuid( ) const private virtual
```

Gets the Guid ID of a close-air-support behavior

◆ **GetSupportObjectID()**

```
virtual int GetSupportObjectID( ) const private virtual
```

Gets the ID of the support object

◆ **GetSupportPosition()**

```
virtual void GetSupportPosition( P3D::DXYZ & lonAltLat ) const private virtual
```

Gets the position for the support (in radians and feet)

◆ **SetSupportObjectID()**

```
virtual void SetSupportObjectID( int objectID ) private virtual
```

Sets the ID of the support object

◆ **SetSupportPosition()**

```
virtual void SetSupportPosition ( const P3D::DXYZ & lonAltLat ) [private] [virtual]
```

Sets the position for the support (in radians and feet)

◆ P3D::IAIBehaviorSearchTrackV01

class P3D::IAIBehaviorSearchTrackV01

Professional Plus Only

Interface for the Search and Track AI Behavior

Inherits IAIBehaviorSearchTrack.

Private Member Functions

```
virtual GUID GetBehaviorGuid () const PURE  
virtual void SetSearchRadius (float radiusFeet) PURE  
virtual float GetSearchRadius () const PURE  
virtual void SetSearchFOVDegrees (float degrees) PURE  
virtual float GetSearchFOVDegrees () const PURE  
virtual void SetTrackRadius (float radiusFeet) PURE  
virtual float GetTrackRadius () const PURE  
virtual void SetTrackFOVDegrees (float degrees) PURE  
virtual float GetTrackFOVDegrees () const PURE  
virtual BOOL IsWithinSearchZone () const PURE  
virtual BOOL IsWithinTrackZone () const PURE
```

Member Function Documentation

◆ GetBehaviorGuid()

```
virtual GUID GetBehaviorGuid ( ) const [private] [virtual]
```

Gets the Guid ID of a search and track behavior

◆ GetSearchFOVDegrees()

```
virtual float GetSearchFOVDegrees ( ) const [private] [virtual]
```

Gets the FOV of a search (in degrees)

◆ GetSearchRadius()

```
virtual float GetSearchRadius ( ) const [private] [virtual]
```

Gets the radius to search (in feet)

◆ GetTrackFOVDegrees()

```
virtual float GetTrackFOVDegrees ( ) const [private] [virtual]
```

Gets the FOV of a track (in degrees)

◆ GetTrackRadius()

```
virtual float GetTrackRadius ( ) const [private] [virtual]
```

Gets the radius to track (in feet)

◆ IsWithinSearchZone()

```
virtual BOOL IsWithinSearchZone ( ) const [private] [virtual]
```

Returns true if behavior is within search zone, false otherwise

◆ IsWithinTrackZone()

```
virtual BOOL IsWithinTrackZone ( ) const [private] [virtual]
```

Returns true if behavior is within track zone, false otherwise

◆ **SetSearchFOVDegrees()**

virtual void SetSearchFOVDegrees (float **degrees**) [private] [virtual]

Sets the FOV of a search (in degrees)

◆ **SetSearchRadius()**

virtual void SetSearchRadius (float **radiusFeet**) [private] [virtual]

Sets the radius to search (in feet)

◆ **SetTrackFOVDegrees()**

virtual void SetTrackFOVDegrees (float **degrees**) [private] [virtual]

Sets the FOV of a track (in degrees)

◆ **SetTrackRadius()**

virtual void SetTrackRadius (float **radiusFeet**) [private] [virtual]

Sets the radius to track (in feet)

◆ **P3D::ISimObjectAIV02**

class P3D::ISimObjectAIV02

The ISimObjectAI interface is an interface on the AI "pilot" implementation for a simobject. A custom AI can be implemented on simobjects created using the ISimObject SDK. The interface may be accessed by systems such as the Traffic Manager or ATC.

Inherits ISimObjectAIV01.

Private Member Functions

```
virtual HRESULT UpdateSimulationFrame ( __in double dDeltaT ) PURE
virtual UNITMODE GetPilotMode () const PURE
virtual void SetPilotMode (UNITMODE eMode, BOOL bOn=TRUE) PURE
virtual void Deactivate () PURE
virtual void Activate () PURE
virtual HRESULT SetWaypoint ( __in BasicWaypoint ) PURE
virtual HRESULT SetDesiredHeading (double dTrueHeading) PURE
virtual HRESULT SetDesiredPitch (double dPitch) PURE
virtual HRESULT SetDesiredSpeed (double dSpeed) PURE
virtual HRESULT SetDesiredAltitude (double dAltitudeMSL) PURE
virtual double GetDesiredHeading () const PURE
virtual double GetDesiredPitch () const PURE
virtual double GetDesiredSpeed () const PURE
virtual double GetDesiredAltitude () const PURE
```

Member Function Documentation

◆ **Activate()**

virtual void Activate () [private] [virtual]

Enables the AI control

◆ **Deactivate()**

virtual void Deactivate () [private] [virtual]

Disables the AI control

◆ **GetDesiredAltitude()**

<code>virtual double GetDesiredAltitude() const</code>	<code>private virtual</code>
Gets the current altitude to be maintained (in feet)	
◆ GetDesiredHeading()	
<code>virtual double GetDesiredHeading() const</code>	<code>private virtual</code>
Gets the current heading to be maintained (in radians)	
◆ GetDesiredPitch()	
<code>virtual double GetDesiredPitch() const</code>	<code>private virtual</code>
Gets the current pitch to be maintained (in radians)	
◆ GetDesiredSpeed()	
<code>virtual double GetDesiredSpeed() const</code>	<code>private virtual</code>
Gets the current speed to be maintained (in feet per second)	
◆ GetPilotMode()	
<code>virtual UNITMODE GetPilotMode() const</code>	<code>private virtual</code>
Returns the current mode of the AI	
◆ SetDesiredAltitude()	
<code>virtual HRESULT SetDesiredAltitude(double dAltitudeMSL)</code>	<code>private virtual</code>
Sets the altitude to be maintained (in feet)	
◆ SetDesiredHeading()	
<code>virtual HRESULT SetDesiredHeading(double dTrueHeading)</code>	<code>private virtual</code>
Sets the heading to be maintained (in radians)	
◆ SetDesiredPitch()	
<code>virtual HRESULT SetDesiredPitch(double dPitch)</code>	<code>private virtual</code>
Sets the pitch angle to be maintained (in radians)	
◆ SetDesiredSpeed()	
<code>virtual HRESULT SetDesiredSpeed(double dSpeed)</code>	<code>private virtual</code>
Sets the desired speed to be maintained (Indicated airspeed for aircraft) (in feet per second)	
◆ SetPilotMode()	
<code>virtual void SetPilotMode(UNITMODE eMode,</code>	
<code>BOOL bOn = TRUE</code>	
<code>)</code>	<code>private virtual</code>
Sets the current mode of the AI	
◆ SetWaypoint()	
<code>virtual HRESULT SetWaypoint(__in BasicWaypoint)</code>	<code>private virtual</code>
Sets the next waypoint for the AI to track to	
◆ UpdateSimulationFrame()	
<code>virtual HRESULT UpdateSimulationFrame(__in double dDeltaT)</code>	<code>private virtual</code>

To be called from the simulation loop to keep the low level AI controllers in sync with the simulation

◆ P3D::IAirplaneAIServiceV02

class P3D::IAirplaneAIServiceV02

The IAirplaneAIService should be implemented on any airplane intended to be controlled by Prepar3D's internal AI Pilot.

Inherits IAirplaneAIServiceV01.

Private Member Functions

```
virtual float GetStallSpeedDirty () const PURE
virtual float GetStallSpeedClean () const PURE
virtual float GetMinDragSpeed () const PURE
virtual float GetZeroLiftAngleOfAttack () const PURE
virtual float GetCriticalAngleOfAttack () const PURE
virtual float GetLinearCLAlpha () const PURE
virtual float GetWingArea () const PURE
virtual float GetWingSpan () const PURE
virtual float GetTotalLongitudinalThrust () const PURE
virtual float GetLiftForce () const PURE
virtual double GetThrottlePercent () const PURE
virtual double GetElevatorPercent () const PURE
virtual double GetAileronPercent () const PURE
virtual double GetRudderPercent () const PURE
virtual double GetSpoilersPercent () const PURE
virtual double GetFlapsPercent () const PURE
virtual void SetThrottlePercent (double dPct) PURE
virtual void SetElevatorPercent (double dPct) PURE
virtual void SetAileronPercent (double dPct) PURE
virtual void SetRudderPercent (double dPct) PURE
virtual void SetFlapsPercent (double dPct) PURE
virtual void SetSpoilersPercent (double dPct) PURE
virtual double CalculateDesiredBank (double dHeadingError, double dDeltaT) PURE
virtual double CalculateDeltaThrottle (double dSpeedError, double dDeltaT) PURE
virtual void SetTaxiHeading (float fHeading) PURE
virtual void SetTaxiSpeed (float fSpeed) PURE
virtual void StopTaxi () PURE
virtual void SetPushBack (BOOL bOn) PURE
virtual void ExtendTailhook () PURE
virtual void RetractTailhook () PURE
virtual float GetTailhookPosition () const PURE
virtual BOOL HasTailhook () const PURE
virtual void ExtendLaunchBar () PURE
virtual void RetractLaunchBar () PURE
virtual float GetLaunchBarPosition () const PURE
virtual BOOL HasLaunchBar () const PURE
virtual void FoldWings () PURE
virtual void UnfoldWings () PURE
virtual float GetLeftWingPosition () const PURE
virtual float GetRightWingPosition () const PURE
virtual float GetFullMilitaryThrottlePosition () const PURE
virtual void ArmNearestCatapult (BOOL bArm) PURE
virtual void FireArmedCatapult () PURE
```

Member Function Documentation

◆ ArmNearestCatapult()

virtual void ArmNearestCatapult (BOOL bArm) [private] [virtual]

Attempts to attach and arm the nearest catapult.

◆ **CalculateDeltaThrottle()**

```
virtual double CalculateDeltaThrottle ( double dSpeedError,  
                                      double dDeltaT  
                                     )
```

private **virtual**

Returns the amount the throttle should be moved this frame based on the current speed error
This is typically calculated with a PID controller based on the airplane and engine dynamics. (-1 -
+1)

◆ **CalculateDesiredBank()**

```
virtual double CalculateDesiredBank ( double dHeadingError,  
                                      double dDeltaT  
                                     )
```

private **virtual**

Returns a desired bank for the AI pilot based on a heading error. This is typically calculated with
a PID controller based on the airplane dynamics (radians)

◆ **ExtendLaunchBar()**

```
virtual void ExtendLaunchBar ( )
```

private **virtual**

Extends the aircraft's launch bar inorder to attach to catapults.

◆ **ExtendTailhook()**

```
virtual void ExtendTailhook ( )
```

private **virtual**

Extends the aircraft's tailhook inorder to catch arrestor cables.

◆ **FireArmedCatapult()**

```
virtual void FireArmedCatapult ( )
```

private **virtual**

Fires the currently armed catapult.

◆ **FoldWings()**

```
virtual void FoldWings ( )
```

private **virtual**

Folds the aircraft's wings, if available.

◆ **GetAileronPercent()**

```
virtual double GetAileronPercent ( ) const
```

private **virtual**

Returns the aileron position (-1 left - +1 right)

◆ **GetCriticalAngleOfAttack()**

```
virtual float GetCriticalAngleOfAttack ( ) const
```

private **virtual**

Returns angle-of-attack at which the aircraft will stall (radians)

◆ **GetElevatorPercent()**

```
virtual double GetElevatorPercent ( ) const
```

private **virtual**

Returns the elevator position (-1 down - +1 up)

◆ **GetFlapsPercent()**

```
virtual double GetFlapsPercent ( ) const
```

private **virtual**

Returns the flap percent deflection (0 - 1)

◆ **GetFullMilitaryThrottlePosition()**

```
virtual float GetFullMilitaryThrottlePosition ( ) const
```

private **virtual**

Returns the throttle position that is considered to be full military power (no afterburner). (position from 0.0 to 1.0)

◆ **GetLaunchBarPosition()**

virtual float GetLaunchBarPosition() const **private** **virtual**

Returns the position of the aircraft's launch bar. (retracted=0.0; extended=1.0)

◆ **GetLeftWingPosition()**

virtual float GetLeftWingPosition() const **private** **virtual**

Returns the current folded position of the aircraft's left wing. (unfolded=0.0; folded=1.0)

◆ **GetLiftForce()**

virtual float GetLiftForce() const **private** **virtual**

Returns the lift force generated by the airplane, including wing, tail, and fuselage) (pounds)

◆ **GetLinearCLAlpha()**

virtual float GetLinearCLAlpha() const **private** **virtual**

Returns the slope of the CL vs. angle-of-attack curve in the linear region, typically between zero lift and the critical angle-of-attack (radians)

◆ **GetMinDragSpeed()**

virtual float GetMinDragSpeed() const **private** **virtual**

Returns speed at which total drag is at its minimum (feet per second)

◆ **GetRightWingPosition()**

virtual float GetRightWingPosition() const **private** **virtual**

Returns the current folded position of the aircraft's right wing. (unfolded=0.0; folded=1.0)

◆ **GetRudderPercent()**

virtual double GetRudderPercent() const **private** **virtual**

Returns rudder percent (-1 left - +1 right)

◆ **GetSpoilersPercent()**

virtual double GetSpoilersPercent() const **private** **virtual**

Return the spoiler position (0 - 1)

◆ **GetStallSpeedClean()**

virtual float GetStallSpeedClean() const **private** **virtual**

Returns stall speed with gear and flaps retracted (feet per second)

◆ **GetStallSpeedDirty()**

virtual float GetStallSpeedDirty() const **private** **virtual**

Returns stall speed with gear and flaps extended (feet per second)

◆ **GetTailhookPosition()**

virtual float GetTailhookPosition() const **private** **virtual**

Returns the position of the aircraft's tailhook. (retracted=0.0; extended=1.0)

◆ GetThrottlePercent()
virtual double GetThrottlePercent() const private virtual
Returns the throttle position (0 - 1)
◆ GetTotalLongitudinalThrust()
virtual float GetTotalLongitudinalThrust() const private virtual
Returns the thrust force in the longitudinal axis (pounds)
◆ GetWingArea()
virtual float GetWingArea() const private virtual
Returns the total area of the main wing (feet squared)
◆ GetWingSpan()
virtual float GetWingSpan() const private virtual
Return the wingspan of the main wing (feet)
◆ GetZeroLiftAngleOfAttack()
virtual float GetZeroLiftAngleOfAttack() const private virtual
Return angle-of-attack at which zero lift is generated (radians)
◆ HasLaunchBar()
virtual BOOL HasLaunchBar() const private virtual
Returns TRUE if the aircraft has a valid launch bar, FALSE otherwise.
◆ HasTailhook()
virtual BOOL HasTailhook() const private virtual
Returns TRUE if the aircraft has a valid tailhook, FALSE otherwise.
◆ RetractLaunchBar()
virtual void RetractLaunchBar() private virtual
Retracts the aircraft's launch bar.
◆ RetractTailhook()
virtual void RetractTailhook() private virtual
Retracts the aircraft's tailhook.
◆ SetAileronPercent()
virtual void SetAileronPercent(double dPct) private virtual
Sets the aileron to a specific position (-1 left - +1 right)
◆ SetElevatorPercent()
virtual void SetElevatorPercent(double dPct) private virtual
Sets the elevator to a specific position (-1 down - +1 up)

◆ **SetFlapsPercent()**

virtual void SetFlapsPercent (double dPct) **private** **virtual**

Sets the flaps to a specific position (0 - +1 extended)

◆ **SetPushBack()**

virtual void SetPushBack (**BOOL** bOn) **private** **virtual**

Triggers the ground handling simulation to move backwards. For example, pushing back from a gate. NOTE: Ground handling is assumed to be tightly coupled and calculated within the airplane simulation.

◆ **SetRudderPercent()**

virtual void SetRudderPercent (double dPct) **private** **virtual**

Sets the rudder to a specific position (-1 left - +1 right)

◆ **SetSpoilersPercent()**

virtual void SetSpoilersPercent (double dPct) **private** **virtual**

Sets the spoilers to a specific position (0 - +1 extended)

◆ **SetTaxiHeading()**

virtual void SetTaxiHeading (float fHeading) **private** **virtual**

Sets the taxi heading (radians) NOTE: Ground handling is assumed to be tightly coupled and calculated within the airplane simulation.

◆ **SetTaxiSpeed()**

virtual void SetTaxiSpeed (float fSpeed) **private** **virtual**

Sets the taxi speed (feet per second) NOTE: Ground handling is assumed to be tightly coupled and calculated within the airplane simulation.

◆ **SetThrottlePercent()**

virtual void SetThrottlePercent (double dPct) **private** **virtual**

Sets the throttle to a specific position (0 - 1)

◆ **StopTaxi()**

virtual void StopTaxi () **private** **virtual**

Triggers the taxi operation to stop. NOTE: Ground handling is assumed to be tightly coupled and calculated within the airplane simulation.

◆ **UnfoldWings()**

virtual void UnfoldWings () **private** **virtual**

Unfolds the aircraft's wings, if available.

◆ **P3D::IHelicopterAIServiceV420**

class P3D::IHelicopterAIServiceV420

Inherits IHelicopterAIServiceV01.

Private Member Functions

virtual double **GetThrottlePercent** () const **PURE**

virtual double **GetCollectivePercent** () const **PURE**

virtual double **GetTorquePedalPercent** () const **PURE**

virtual double **GetCyclicLateralPercent** () const **PURE**

```
virtual double GetCyclicLateralPercent() const PURE
virtual void SetThrottlePercent (double dPct) PURE
virtual void SetCollectivePercent (double dPct) PURE
virtual void SetTorquePedalPercent (double dPct) PURE
virtual void SetCyclicLateralPercent (double dPct) PURE
virtual void SetCyclicLongitudinalPercent (double dPct) PURE
```

Member Function Documentation

◆ **GetCollectivePercent()**

```
virtual double GetCollectivePercent ( ) const private virtual
```

Returns the collective position (0 - 1)

◆ **GetCyclicLateralPercent()**

```
virtual double GetCyclicLateralPercent ( ) const private virtual
```

Returns the cyclic L/R position (0 - 1)

◆ **GetCyclicLongitudinalPercent()**

```
virtual double GetCyclicLongitudinalPercent ( ) const private virtual
```

Returns the cyclic F/A position (0 - 1)

◆ **GetThrottlePercent()**

```
virtual double GetThrottlePercent ( ) const private virtual
```

Returns the throttle position (0 - 1)

◆ **GetTorquePedalPercent()**

```
virtual double GetTorquePedalPercent ( ) const private virtual
```

Returns the torque pedals position (-1 - 1)

◆ **SetCollectivePercent()**

```
virtual void SetCollectivePercent ( double dPct ) private virtual
```

Sets the collective to a specific position (0 - 1)

◆ **SetCyclicLateralPercent()**

```
virtual void SetCyclicLateralPercent ( double dPct ) private virtual
```

Sets the cyclic L/R position (0 - 1)

◆ **SetCyclicLongitudinalPercent()**

```
virtual void SetCyclicLongitudinalPercent ( double dPct ) private virtual
```

Sets the cyclic F/A position (0 - 1)

◆ **SetThrottlePercent()**

```
virtual void SetThrottlePercent ( double dPct ) private virtual
```

Sets the throttle to a specific position (0 - 1)

◆ **SetTorquePedalPercent()**

```
virtual void SetTorquePedalPercent ( double dPct ) private virtual
```

Sets the torque pedals to a specific position (-1 - 1)

◆ P3D::IGroundVehicleAIServiceV01

class P3D::IGroundVehicleAIServiceV01

This interface enables ground vehicle implementations to be utilized by Prepar3D's internal AI controllers.

Inherits IGroundVehicleAIService.

◆ P3D::IWeaponsSystemV440

class P3D::IWeaponsSystemV440

Professional Plus Only

Interface to the Prepar3D native weapon system. Can also be used to implement a custom weapon system

Inherits IWeaponsSystemV430.

Private Member Functions

```
virtual BOOL GetIgnoreAttachmentForces () const PURE
virtual void SetIgnoreAttachmentForces (BOOL enabled) PURE
virtual BOOL GetIgnoreAttachmentWeight () const PURE
virtual void SetIgnoreAttachmentWeight (BOOL enabled) PURE
virtual UINT GetNumberOfStations () const PURE
virtual UINT GetStationQuantity (UINT iStationIndex) const PURE
virtual BOOL HasPylon (UINT iStationIndex) const PURE
virtual UINT GetNumberOfPylonPoints (UINT iStationIndex) const PURE
virtual HRESULT GetWeapon ( __in UINT iStationIndex, __in UINT iPylonIndex, __out UINT &uObjectld, __out IWeaponServiceV400 **ppWeapon) const PURE
virtual HRESULT GetPylon ( __in UINT iStationIndex, __out UINT &uObjectld, __out IPylonService **ppPylon) const PURE
virtual void SetStationLoadOut ( __in UINT32 stationOut, __in LPCTSTR pszWeaponTitle, __in UINT32 roundsRemaining, __in UINT32 roundsDefault, __in LPCTSTR pszPylonTitle) PURE
virtual BOOL IsSystemOn () const PURE
virtual BOOL IsSystemArmed () const PURE
virtual BOOL IsSafetyOn () const PURE
virtual void ToggleSystem () PURE
virtual void ToggleArmed () PURE
virtual void ToggleSafety () PURE
virtual void EngageTrigger (BOOL bSingleShot) PURE
virtual void DisengageTrigger () PURE
virtual void TriggerJettison () PURE
virtual BOOL IsStationSelected (UINT iStationIndex) const PURE
virtual void SelectNextStation () PURE
virtual void SelectPreviousStation () PURE
virtual void SetSelectedWeaponTypeIndex (UINT uData) PURE
virtual void ToggleStation (UINT iStationIndex) PURE
virtual void SelectStationOn (UINT iStationIndex, BOOL bExclusiveOn) PURE
virtual void SelectStationOff (UINT iStationIndex, BOOL bAllOff) PURE
virtual void SelectPylonPointOn (UINT iStationIndex, UINT iPylonPoint, BOOL bExclusiveOn) PURE
virtual void SelectPylonPointOff (UINT iStationIndex, UINT iPylonPoint, BOOL bAllOff) PURE
virtual void SelectNextWeapon () PURE
virtual void SelectPreviousWeapon () PURE
virtual void ResetWeapons () PURE
```

Member Function Documentation

◆ DisengageTrigger()

virtual void DisengageTrigger () [private] [virtual]

Disengages a weapon system trigger

◆ EngageTrigger()

```
virtual void EngageTrigger( BOOL bSingleShot ) private virtual
```

Engages a weapon system trigger

Parameters

bSingleShot Sets max of one shot per trigger engage

Note

In order to work properly, [EngageTrigger\(\)](#) cannot be called every frame. At least one frame must pass without a call to [EngageTrigger\(\)](#) to allow the trigger state to reset.

◆ [GetIgnoreAttachmentForces\(\)](#)

```
virtual BOOL GetIgnoreAttachmentForces( ) const private virtual
```

Gets the Global Attachment Setting for ignoring forces

◆ [GetIgnoreAttachmentWeight\(\)](#)

```
virtual BOOL GetIgnoreAttachmentWeight( ) const private virtual
```

Gets the Global Attachment Setting for ignoring weight

◆ [GetNumberOfPylonPoints\(\)](#)

```
virtual UINT GetNumberOfPylonPoints( UINT iStationIndex ) const private virtual
```

Gets the total number of pylon points that a station contains

◆ [GetNumberOfStations\(\)](#)

```
virtual UINT GetNumberOfStations( ) const private virtual
```

Gets the number of stations available to the weapon system

◆ [GetPylon\(\)](#)

```
virtual HRESULT GetPylon( _in UINT iStationIndex,  
                         _out UINT & uObjectId,  
                         _out IPylonService** ppPylon  
                     ) const private virtual
```

Gets a pylon

◆ [GetStationQuantity\(\)](#)

```
virtual UINT GetStationQuantity( UINT iStationIndex ) const private virtual
```

Gets the the total number of weapons loaded at the given

◆ [GetWeapon\(\)](#)

```
virtual HRESULT GetWeapon( _in UINT iStationIndex,  
                           _in UINT iPylonIndex,  
                           _out UINT & uObjectId,  
                           _out IWeaponServiceV400** ppWeapon  
                     ) const private virtual
```

Gets a weapon

◆ [HasPylon\(\)](#)

```
virtual BOOL HasPylon( UINT iStationIndex ) const private virtual
```

Returns true if a station contains a pylon, false otherwise

◆ [IsSafetyOn\(\)](#)

```
virtual BOOL IsSafetyOn( ) const private virtual
```

Returns whether or not system's safety is on.

◆ IsStationSelected()

virtual **BOOL** IsStationSelected (**UINT** iStationIndex) const **private** **virtual**

Returns true if a station is selected

◆ IsSystemArmed()

virtual BOOL IsSystemArmed() const

Returns whether or not the system is armed.

◆ IsSystemOn()

virtual BOOL IsSystemOn() const

Returns whether or not the system is on.

◆ ResetWeapons()

virtual void ResetWeapons()

Reset the weapon loadouts to their original state. This does not change station/pylon point selection.

◆ SelectNextStation()

virtual void SelectNextStation()

Selects the next station, even if empty

◆ SelectNextWeapon()

virtual void SelectNextWeapon()

Select the next available weapon regardless of type

◆ SelectPreviousStation()

virtual void SelectPreviousStation ()

Selects the previous station, even if empty

◆ SelectPreviousWeapon()

virtual void SelectPreviousWeapon()

Select the previous available weapon regardless of type

◆ SelectPylonPointOff()

```
virtual void SelectPylonPointOff ( UINT iStationIndex,  
                                  UINT iPylonPoint,  
                                  BOOL bAllOff  
 )
```

Turns selection pylon point off

Parameters

bAllOff Turns all pylon points off at station

◆ SelectPylonPointOn()

Turns selection pylon point on

Parameters

bExclusiveOn Turns all other pylon points off at station

◆ **SelectStationOff()**

```
virtual void SelectStationOff ( UINT iStationIndex,  
                                BOOL bAllOff  
                            )
```

private **virtual**

Turns selected station off

Parameters

bAllOff Turns all stations off

◆ **SelectStationOn()**

```
virtual void SelectStationOn ( UINT iStationIndex,  
                               BOOL bExclusiveOn  
                           )
```

private **virtual**

Turns selected station on

Parameters

bExclusiveOn Turns all other stations off

◆ **SetIgnoreAttachmentForces()**

```
virtual void SetIgnoreAttachmentForces ( BOOL enabled )
```

private **virtual**

Sets the Global Attachment Setting for ignoring forces

◆ **SetIgnoreAttachmentWeight()**

```
virtual void SetIgnoreAttachmentWeight ( BOOL enabled )
```

private **virtual**

Sets the Global Attachment Setting for ignoring forces

◆ **SetSelectedWeaponTypeIndex()**

```
virtual void SetSelectedWeaponTypeIndex ( UINT uData )
```

private **virtual**

Selects the next station for the weapon type corresponding to the index defined in the WeaponSelectorTypes list in attachments.xml

◆ **SetStationLoadOut()**

```
virtual void SetStationLoadOut ( _in UINT32 stationIndex,  
                                 _in LPCTSTR pszWeaponTitle,  
                                 _in UINT32 roundsRemaining,  
                                 _in UINT32 roundsDefault,  
                                 _in LPCTSTR pszPylonTitle  
                            )
```

private **virtual**

Sets the station weapon and pylon

◆ **ToggleArmed()**

```
virtual void ToggleArmed ( )
```

private **virtual**

Arms and disarms a weapon system

◆ **ToggleSafety()**

```
virtual void ToggleSafety ( )
```

private **virtual**

Toggles a weapon system safety on and off

◆ **ToggleStation()**

```
virtual void ToggleStation ( UINT iStationIndex )
```

private **virtual**

Toggles the selection of the given station on and off

◆ **ToggleSystem()**

```
virtual void ToggleSystem( ) [private] [virtual]
```

Toggles a weapon system on and off

◆ **TriggerJettison()**

```
virtual void TriggerJettison( ) [private] [virtual]
```

Triggers jettison of a currently selected weapon system

Note
In order to work properly, [TriggerJettison\(\)](#) cannot be called every frame. At least one frame must pass without a call to [TriggerJettison\(\)](#) to allow the trigger state to reset.

◆ **P3D::IWeaponServiceV420**

```
class P3D::IWeaponServiceV420
```

Professional Plus Only

Interface for getting weapon parameters for this object

Inherits IWeaponServiceV400.

Private Member Functions

```
virtual HRESULT SetIsAttachedToOwner (BOOL bAttached, UINT uOwnerId, BOOL bJettisoned) PURE
virtual BOOL IsAttachedToOwner () const PURE
virtual UINT GetOwnerId () const PURE
virtual HRESULT GetAttachOffsetFeet ( __out P3D::DXYZ &vOffset) const PURE
virtual BOOL CanWeaponBeReleased () const PURE
virtual float GetAerodynamicsDragCoefficient (float fMach) const PURE
virtual HRESULT GetType ( __out LPWSTR pszType, __in unsigned int uLength) const PURE
virtual BOOL GetCausesWeaponCollision () const PURE
```

Member Function Documentation

◆ **CanWeaponBeReleased()**

```
virtual BOOL CanWeaponBeReleased ( ) const [private] [virtual]
```

Called upon firing of weapon. The weapon implementation can block being released

◆ **GetAerodynamicsDragCoefficient()**

```
virtual float GetAerodynamicsDragCoefficient ( float fMach ) const [private] [virtual]
```

Gets the aerodynamic drag for the weapon loadout UI in SimDirector

◆ **GetAttachOffsetFeet()**

```
virtual HRESULT GetAttachOffsetFeet ( __out P3D::DXYZ & vOffset ) const [private] [virtual]
```

Gets the offset on the weapon in which it is attached to the parent

◆ **GetCausesWeaponCollision()**

```
virtual BOOL GetCausesWeaponCollision ( ) const [private] [virtual]
```

Gets whether or not the weapon should collide with other weapons

◆ **GetOwnerId()**

```
virtual UINT GetOwnerId ( ) const [private] [virtual]
```

ID of object in which weapon is attached (should remain valid even after detached)

◆ GetType()

```
virtual HRESULT GetType ( __out LPWSTR pszType,
                        __in unsigned int uLength
                      ) const private virtual
```

Gets the string type of weapon (e.g. "AAM", "SAM"). These are defined for native weapons in sim.cfg. It is dependent on the weapon implementation, but can be used for arbitrary categorization

◆ IsAttachedToOwner()

```
virtual BOOL IsAttachedToOwner ( ) const private virtual
```

Is weapon currently attached to parent object

◆ SetIsAttachedToOwner()

```
virtual HRESULT SetIsAttachedToOwner ( BOOL bAttached,
                                       UINT uOwnerId,
                                       BOOL bJettisoned
                                     ) private virtual
```

Called from weapon system when attached, jettisoned, or fired (0 = invalid id)

◆ P3D::ICountermeasureSystemV01

class P3D::ICountermeasureSystemV01

Professional Plus Only

Used to implement or query for a countermeasure system

Inherits ICountermeasureSystem.

Private Member Functions

```
virtual UINT GetNumberOfStations () const PURE
virtual UINT GetStationQuantity (UINT iStationIndex) const PURE
virtual HRESULT GetCountermeasure ( __in UINT iStationIndex, __in UINT iPylonIndex,
                                    __out UINT &uObjectId, __out ICountermeasureService **ppCM) const
                                    PURE
virtual BOOL IsSystemOn () const PURE
virtual BOOL IsSystemArmed () const PURE
virtual void ToggleSystem () PURE
virtual void ToggleArmed () PURE
virtual void EngageTrigger (BOOL bSingleShot) PURE
virtual void DisengageTrigger () PURE
virtual BOOL IsStationSelected (UINT iStationIndex) const PURE
virtual void SelectNextStation () PURE
virtual void SelectPreviousStation () PURE
virtual void ToggleStation (UINT iStationIndex) PURE
virtual void SelectStationOn (UINT iStationIndex, BOOL bExclusiveOn) PURE
virtual void SelectStationOff (UINT iStationIndex, BOOL bAllOff) PURE
virtual void SelectNextCountermeasure () PURE
virtual void SelectPreviousCountermeasure () PURE
virtual void ResetCountermeasures () PURE
```

Member Function Documentation

◆ DisengageTrigger()

```
virtual void DisengageTrigger ( ) private virtual
```

Detriggers a countermeasure

◆ EngageTrigger()

```
virtual void EngageTrigger ( BOOL bSingleShot ) private virtual
```

Triggers a countermeasure

Parameters

bSingleShot Sets max of one shot per trigger engage

◆ **GetCountermeasure()**

```
virtual HRESULT GetCountermeasure( __in UINT iStationIndex,  
                                 __in UINT iPylonIndex,  
                                 __out UINT & uObjectId,  
                                 __out ICountermeasureService ** ppCM  
                               ) const [private] [virtual]
```

Gets a countermeasure system

◆ **GetNumberOfStations()**

```
virtual UINT GetNumberOfStations( ) const [private] [virtual]
```

Gets number of countermeasure stations

◆ **GetStationQuantity()**

```
virtual UINT GetStationQuantity( UINT iStationIndex ) const [private] [virtual]
```

◆ **IsStationSelected()**

```
virtual BOOL IsStationSelected( UINT iStationIndex ) const [private] [virtual]
```

Returns true if a station is selected

◆ **IsSystemArmed()**

```
virtual BOOL IsSystemArmed( ) const [private] [virtual]
```

Returns whether or not the system is armed.

◆ **IsSystemOn()**

```
virtual BOOL IsSystemOn( ) const [private] [virtual]
```

Returns whether or not the system is on.

◆ **ResetCountermeasures()**

```
virtual void ResetCountermeasures( ) [private] [virtual]
```

Reset all countermeasures

◆ **SelectNextCountermeasure()**

```
virtual void SelectNextCountermeasure( ) [private] [virtual]
```

Select the next countermeasure

◆ **SelectNextStation()**

```
virtual void SelectNextStation( ) [private] [virtual]
```

Selects the next station

◆ **SelectPreviousCountermeasure()**

```
virtual void SelectPreviousCountermeasure( ) [private] [virtual]
```

Select the previous countermeasure

◆ **SelectPreviousStation()**

```
virtual void SelectPreviousStation( ) [private] [virtual]
```

Selects the previous station

◆ **SelectStationOff()**

```
virtual void SelectStationOff( UINT iStationIndex,  
                           BOOL bAllOff  
                           )
```

private **virtual**

Turns selected station off

Parameters

bAllOff Turns all stations off

◆ **SelectStationOn()**

```
virtual void SelectStationOn( UINT iStationIndex,  
                           BOOL bExclusiveOn  
                           )
```

private **virtual**

Turns selected station on

Parameters

bExclusiveOn Turns all other stations off

◆ **ToggleArmed()**

```
virtual void ToggleArmed( )
```

private **virtual**

Arms and disarms a countermeasure system

◆ **ToggleStation()**

```
virtual void ToggleStation( UINT iStationIndex )
```

private **virtual**

Toggles a specific station on and off

◆ **ToggleSystem()**

```
virtual void ToggleSystem( )
```

private **virtual**

Toggles a countermeasure system on and off

◆ **P3D::ICountermeasureServiceV02**

class P3D::ICountermeasureServiceV02

Professional Plus Only

Interface for getting countermeasure parameters for this object

Inherits ICountermeasureServiceV01.

Private Member Functions

```
virtual HRESULT SetIsAttachedToOwner( BOOL bAttached, UINT uOwnerId) PURE
```

```
virtual BOOL IsAttachedToOwner() const PURE
```

```
virtual UINT GetOwnerId() const PURE
```

```
virtual HRESULT GetAttachOffsetFeet( __out P3D::DXYZ &vOffset) const PURE
```

```
virtual BOOL GetCausesWeaponCollision() const PURE
```

Member Function Documentation

◆ **GetAttachOffsetFeet()**

```
virtual HRESULT GetAttachOffsetFeet( __out P3D::DXYZ &vOffset) const
```

private **virtual**

Gets the offset on the weapon in which it is attached to the parent

◆ **GetCausesWeaponCollision()**

```
virtual BOOL GetCausesWeaponCollision( ) const
```

private **virtual**

Gets whether or not the countermeasure should collide with weapons

◆ **GetOwnerId()**

```
virtual UINT GetOwnerId ( ) const [private] [virtual]
```

ID of object in which countermeasure is attached (should remain valid even after detached)

◆ **IsAttachedToOwner()**

```
virtual BOOL IsAttachedToOwner ( ) const [private] [virtual]
```

Is weapon currently attached to parent object

◆ **SetIsAttachedToOwner()**

```
virtual HRESULT SetIsAttachedToOwner ( BOOL bAttached,
                                      UINT uOwnerId
                                    ) [private] [virtual]
```

Called from countermeasure system when attached, jettisoned, or fired (0 = invalid id)

◆ P3D::IGunSystemV440

class P3D::IGunSystemV440

Professional Plus Only

Interface for getting gun system parameters for this object

Inherits IGunSystemV400.

Private Member Functions

```

virtual BOOL AddGun ( __in IGunV400 *pGun, __in UINT stationIndex) PURE
virtual BOOL RemoveGun ( __in UINT stationIndex) PURE
virtual const IGunV400 * GetGun ( __in UINT stationIndex) const PURE
    virtual void SetIsActive ( __in BOOL isActive) PURE
    virtual BOOL GetIsActive () const PURE
    virtual BOOL GetIsFiring () const PURE
    virtual void SetIsAutomatedGunsEnabled ( __in BOOL
                                           isAutomatedGunsEnabled) PURE
    virtual BOOL GetIsAutomatedGunsEnabled () const PURE
    virtual void EngageTrigger () PURE
    virtual void DisengageTrigger () PURE
    virtual UINT GetNumberOfStations () const PURE
    virtual UINT GetNumberOfGuns () const PURE
    virtual BOOL IsStationIndexValid ( __in UINT stationIndex) const PURE
    virtual BOOL GetIsGunPresentAtStation ( __in UINT stationIndex) const PURE
    virtual BOOL GetIsGunSelectedAtStation ( __in UINT stationIndex) const PURE
    virtual BOOL IsSystemOn () const PURE
    virtual void ToggleSystem () PURE
    virtual void ToggleStation ( __in UINT stationIndex) PURE
    virtual void SelectStationOn (UINT iStationIndex, BOOL bExclusiveOn) PURE
    virtual void SelectStationOff (UINT iStationIndex, BOOL bAllOff) PURE
    virtual void CreateTracerEffect ( __in __notnull const WCHAR *pszEffectName,
                                     __in const DXYZ *pvLonAltLat, __in const DXYZ *pvPHB, __out void
                                     **ppEffect) PURE
    virtual void MoveTracerEffect ( __in const DXYZ *pvLonAltLat, __in const DXYZ
                                   *pvPHB, __in void *pEffect) PURE
    virtual void DestroyTracerEffect ( __in void *pEffect) PURE
    virtual BOOL CheckBulletCollision ( __in const DXYZ *pvLonAltLat, __in const DXYZ
                                       *pvDeltaOffset, __out COLLISIONTYPE &eCollision, __out
                                       IUnknown **ppUnkObjectHit) PURE
    virtual void ToggleAutomaticGuns () PURE
    virtual void SetPitchPercent (float fPercent) PURE
    virtual void SetHeadingPercent (float fPercent) PURE
    virtual void IncrementLeft () PURE
    virtual void IncrementRight () PURE
    virtual void IncrementUp () PURE

```

```
virtual void IncrementDown () PURE  
virtual void IncrementLeftAndUp () PURE  
virtual void IncrementLeftAndDown () PURE  
virtual void IncrementRightAndUp () PURE  
virtual void IncrementRightAndDown () PURE  
virtual void ResetGuns () PURE  
virtual void SetCrosshairTarget ( __in const double &lat, __in const double &lon,  
                                __in const double &alt) PURE  
virtual void ClearCrosshairTarget () PURE
```

Member Function Documentation

◆ AddGun()

```
virtual BOOL AddGun ( __in IGunV400 * pGun,  
                      __in UINT stationIndex  
                    )
```

private **virtual**

Adds an IGun implementation to the internal GunSystem at the given station. Returns TRUE if successfully added, and FALSE otherwise. If successfully added, the IGun reference count will be increased by one.

◆ CheckBulletCollision()

```
virtual BOOL CheckBulletCollision ( __in const DXYZ * pvLonAltLat,  
                                   __in const DXYZ * pvDeltaOffset,  
                                   __out COLLISIONTYPE & eCollision,  
                                   __out IUnknown ** ppUnkObjectHit  
                                 )
```

private **virtual**

◆ ClearCrosshairTarget()

```
virtual void ClearCrosshairTarget ( )
```

private **virtual**

◆ CreateTracerEffect()

```
virtual void CreateTracerEffect ( __in __notnull const WCHAR * pszEffectName,  
                                 __in const DXYZ * pvLonAltLat,  
                                 __in const DXYZ * pvPHB,  
                                 __out void ** ppEffect  
                               )
```

private **virtual**

Creates a tracer visual effect at the given location. Longitude, altitude, and latitude units are radians and feet. Pitch, heading, and bank units are radians

◆ DestroyTracerEffect()

```
virtual void DestroyTracerEffect ( __in void * pEffect )
```

private **virtual**

Removes a tracer effect if it is still alive. Performs collision detection at a location in the form longitude, altitude, and latitude, where the units are radians and feet. The delta offset should be given from this location in feet

◆ DisengageTrigger()

```
virtual void DisengageTrigger ( )
```

private **virtual**

Disengages the trigger of a gun system

◆ EngageTrigger()

```
virtual void EngageTrigger ( )
```

private **virtual**

Engages the trigger of a gun system

◆ GetGun()

```
virtual const IGunV400* GetGun ( __in UINT stationIndex ) const
```

private **virtual**

Gets a pointer to an IGun implementation at the given station if it exists, otherwise returns NULL.

◆ **GetIsActive()**
virtual **BOOL** GetIsActive () const **private** **virtual**

Gets the active state of the gun system

◆ **GetIsAutomatedGunsEnabled()**
virtual **BOOL** GetIsAutomatedGunsEnabled () const **private** **virtual**

Gets the state of the automatic gun targeting.

◆ **GetIsFiring()**
virtual **BOOL** GetIsFiring () const **private** **virtual**

Gets if a gun system is currently firing

◆ **GetIsGunPresentAtStation()**
virtual **BOOL** GetIsGunPresentAtStation (__in **UINT** stationIndex) const **private** **virtual**

Checks if a station contains a gun

◆ **GetIsGunSelectedAtStation()**
virtual **BOOL** GetIsGunSelectedAtStation (__in **UINT** stationIndex) const **private** **virtual**

Checks if a station contains a selected gun

◆ **GetNumberOfGuns()**
virtual **UINT** GetNumberOfGuns () const **private** **virtual**

Returns the number of guns loaded by the gun system

◆ **GetNumberOfStations()**
virtual **UINT** GetNumberOfStations () const **private** **virtual**

Returns the number of stations available to the gun system

◆ **IncrementDown()**
virtual **void** IncrementDown () **private** **virtual**

Moves gun down

◆ **IncrementLeft()**
virtual **void** IncrementLeft () **private** **virtual**

Moves gun left

◆ **IncrementLeftAndDown()**
virtual **void** IncrementLeftAndDown () **private** **virtual**

Moves gun left and down

◆ **IncrementLeftAndUp()**
virtual **void** IncrementLeftAndUp () **private** **virtual**

Moves gun left and up

◆ **IncrementRight()**

<code>virtual void IncrementRight()</code>	<code>private virtual</code>
Moves gun right	
◆ IncrementRightAndDown()	
<code>virtual void IncrementRightAndDown()</code>	<code>private virtual</code>
Moves gun right and down	
◆ IncrementRightAndUp()	
<code>virtual void IncrementRightAndUp()</code>	<code>private virtual</code>
Moves gun right and up	
◆ IncrementUp()	
<code>virtual void IncrementUp()</code>	<code>private virtual</code>
Moves gun up	
◆ IsStationIndexValid()	
<code>virtual BOOL IsStationIndexValid(__in UINT stationIndex) const</code>	<code>private virtual</code>
Checks if an index refers to an existing station	
◆ IsSystemOn()	
<code>virtual BOOL IsSystemOn() const</code>	<code>private virtual</code>
Returns whether or not the system is on.	
◆ MoveTracerEffect()	
<code>virtual void MoveTracerEffect(__in const DXYZ * pvLonAltLat,</code>	
<code> __in const DXYZ * pvPHB,</code>	
<code> __in void * pEffect</code>	
<code>)</code>	<code>private virtual</code>
Moves an existing visual effect to the given location. Same position and rotation values above apply	
◆ RemoveGun()	
<code>virtual BOOL RemoveGun(__in UINT stationIndex)</code>	<code>private virtual</code>
Removes an IGun implementation from the internal GunSystem at the given station. Returns TRUE if successfully removed, and FALSE otherwise. If successfully removed, the IGun reference count will be decreased by one.	
◆ ResetGuns()	
<code>virtual void ResetGuns()</code>	<code>private virtual</code>
Resets the gun loadout to its original state. This does not change station/pylon point selection.	
◆ SelectStationOff()	
<code>virtual void SelectStationOff(UINT iStationIndex,</code>	
<code> BOOL bAllOff</code>	
<code>)</code>	<code>private virtual</code>
Turns selected station off	
Parameters	
bAllOff	Turns all stations off
◆ SelectStationOn()	

```
virtual void SelectStationOn ( UINT iStationIndex,  
    BOOL bExclusiveOn  
 )
```

private **virtual**

Turns selected station on

Parameters

bExclusiveOn Turns all other stations off

◆ **SetCrosshairTarget()**

```
virtual void SetCrosshairTarget ( __in const double & lat,  
    __in const double & lon,  
    __in const double & alt  
 )
```

private **virtual**

◆ **SetHeadingPercent()**

```
virtual void SetHeadingPercent ( float fPercent )
```

private **virtual**

Change the heading of guns based on percentage. [-1, 1]

◆ **SetIsActive()**

```
virtual void SetIsActive ( __in BOOL isActive )
```

private **virtual**

Sets a gun system to active (true) or inactive (false)

◆ **SetIsAutomatedGunsEnabled()**

```
virtual void SetIsAutomatedGunsEnabled ( __in BOOL isAutomatedGunsEnabled )
```

private **virtual**

Sets the state of the automatic gun targeting.

◆ **SetPitchPercent()**

```
virtual void SetPitchPercent ( float fPercent )
```

private **virtual**

Change the pitch of guns based on percentage. [-1, 1]

◆ **ToggleAutomaticGuns()**

```
virtual void ToggleAutomaticGuns ( )
```

private **virtual**

Toggles the state of automatic gun targeting on and off

◆ **ToggleStation()**

```
virtual void ToggleStation ( __in UINT stationIndex )
```

private **virtual**

Toggles a station on and off

◆ **ToggleSystem()**

```
virtual void ToggleSystem ( )
```

private **virtual**

Toggles a gun system on and off

◆ **P3D::IGunV400**

class P3D::IGunV400

Professional Plus Only

Interface for getting gun parameters for this object

Inherits IUnknown.

Private Member Functions

```
virtual void Simulate ( __in double deltaT )
```

```
virtual void Fire( __in double deltaT ) PURE
virtual void Purge() PURE
virtual void Stop() PURE
virtual void SetRoundsRemaining( __in uint ammoCount ) PURE
virtual uint GetRoundsRemaining() const PURE
virtual void ResetRounds() PURE
virtual const WCHAR* GetName() const PURE
virtual const WCHAR* GetGunType() const PURE
virtual void Rotate( __in double xAxisOffset, __in double yAxisOffset, __in double
deltaT ) PURE
virtual void ProcessTargeting( __in const P3D::DXYZ& targetLla, __in const
P3D::DXYZ& targetBodyVelocity, __in const P3D::DXYZ& targetBodyAcceleration,
__in const P3D::DXYZ& targetOrientation,
__in double deltaT ) PURE
```

Member Function Documentation

◆ Fire()

```
virtual HRESULT Fire( __in double deltaT ) [private] [virtual]
```

Called once per step on selected guns. **Fire()** will be repeatedly called while the trigger is engaged. Users can use an HRESULT return type

◆ GetGunType()

```
virtual const WCHAR* GetGunType() const [private] [virtual]
```

Gets the type of a gun

◆ GetName()

```
virtual const WCHAR* GetName() const [private] [virtual]
```

Gets the name of a gun

◆ GetRoundsRemaining()

```
virtual uint GetRoundsRemaining() const [private] [virtual]
```

Gets the total number of rounds in a gun

◆ ProcessTargeting()

```
virtual void ProcessTargeting(
    __in const P3D::DXYZ& targetLla,
    __in const P3D::DXYZ& targetBodyVelocity,
    __in const P3D::DXYZ& targetBodyAcceleration,
    __in const P3D::DXYZ& targetOrientation,
    __in double deltaT
) [private] [virtual]
```

called if automatic guns are enabled, providing the developer with information on the target. Target position is longitude, altitude and latitude in that order, where the units are radians and feet. Target velocity and acceleration are in feet per second while orientation is in world coordinate radians.

◆ Purge()

```
virtual void Purge() [private] [virtual]
```

Called on all guns when the trigger is released

◆ ResetRounds()

```
virtual void ResetRounds() [private] [virtual]
```

Resets the total number of rounds in a gun. Called on each gun when ResetGuns() is called by the GunSystem.

◆ Rotate()

```
virtual void Rotate ( __in double xAxisOffset,  
                     __in double yAxisOffset,  
                     __in double deltaT  
)
```

private virtual

Called when the user provides input to rotate guns. Values passed should be expected to be mapped from -1.0 to 1.0.

◆ SetRoundsRemaining()

```
virtual void SetRoundsRemaining ( __in UINT ammoCount ) private virtual
```

Sets the total number of rounds in a gun

◆ Simulate()

```
virtual void Simulate ( __in double deltaT ) private virtual
```

Called once per step on all guns. Delta time is in seconds

◆ Stop()

```
virtual void Stop ( ) private virtual
```

Called on all guns while there are no user inputs for gun rotations and automatic targeting is disabled

◆ P3D::IFireControlSystemV01

class P3D::IFireControlSystemV01

Professional Plus Only

Interface for getting fire control system parameters for this object

Inherits IFireControlSystem.

Private Member Functions

```
virtual UINT GetSelectedTargetID () const PURE  
virtual void SetSelectedTargetID (UINT id) PURE  
virtual HRESULT GetSelectedTargetMissionID ( __out GUID &guid ) const PURE  
virtual HRESULT SetSelectedTargetMissionID ( __in const GUID &guid ) PURE  
virtual BOOL GetTargetLLA ( __out P3D::DXYZ &vLLA ) const PURE  
virtual void SetTargetLLA ( __in const P3D::DXYZ &vLLA ) PURE
```

Member Function Documentation

◆ GetSelectedTargetID()

```
virtual UINT GetSelectedTargetID ( ) const private virtual
```

Get the ID of the target selected by the fire control system

◆ GetSelectedTargetMissionID()

```
virtual HRESULT GetSelectedTargetMissionID ( __out GUID & guid ) const private virtual
```

Gets the instance ID of the target selected by the fire control system (Structured scenarios with objects only)

◆ GetTargetLLA()

```
virtual BOOL GetTargetLLA ( __out P3D::DXYZ & vLLA ) const private virtual
```

If the fire control system's target is a latitude/longitude/altitude, this will return that position. Otherwise the return will be FALSE. (radians/radians/feet)

◆ SetSelectedTargetID()

```
virtual void SetSelectedTargetID ( UINT id ) [private] [virtual]
```

Sets the fire control system target by object

◆ SetSelectedTargetMissionID()

```
virtual HRESULT SetSelectedTargetMissionID ( __in const GUID & guid ) [private] [virtual]
```

Sets the fire control system target by instance ID (Missions only)

◆ SetTargetLLA()

```
virtual void SetTargetLLA ( __in const P3D::DXYZ & vLLA ) [private] [virtual]
```

Sets the fire control system's target to be a latitude/longitude/altitude. (radians/radians/feet)

◆ P3D::IGuidanceSystemV01

class P3D::IGuidanceSystemV01

Professional Plus Only

Interface for getting guidance parameters for this object

Inherits IGuidanceSystem.

Private Member Functions

```
virtual void SetTargetObjectID ( UINT targetedObjectID ) PURE
```

```
virtual UINT GetTargetObjectID () const PURE
```

```
virtual void SetTargetLLA ( __in const P3D::DXYZ &vLLA ) PURE
```

```
virtual BOOL GetTargetLLA ( __out P3D::DXYZ &vLLA ) const PURE
```

Member Function Documentation

◆ GetTargetLLA()

```
virtual BOOL GetTargetLLA ( __out P3D::DXYZ & vLLA ) const [private] [virtual]
```

If the guidance system's target is a latitude/longitude/altitude, this will return that position.
Otherwise the return will be FALSE. (radians/radians/feet)

◆ GetTargetObjectID()

```
virtual UINT GetTargetObjectID ( ) const [private] [virtual]
```

Gets the ID of a target

◆ SetTargetLLA()

```
virtual void SetTargetLLA ( __in const P3D::DXYZ & vLLA ) [private] [virtual]
```

sets the guidance system's target to be a latitude/longitude/altitude. (radians/radians/feet)

◆ SetTargetObjectID()

```
virtual void SetTargetObjectID ( UINT targetedObjectID ) [private] [virtual]
```

Sets the ID of a target

◆ P3D::IPylonServiceV01

class P3D::IPylonServiceV01

Professional Plus Only

Interface for getting parameters for a weapon pylon

Inherits IPylonService.

Private Member Functions

```
virtual HRESULT SetOwnerID ( __in uint uiOwnerID ) PURE
```

```

virtual HRESULT GetOwnerId( _in uint ownerId ) const PURE
virtual uint GetOwnerId() const PURE
virtual HRESULT GetAttachOffsetFeet( _out P3D::DXYZ &vOffset) const PURE

```

Member Function Documentation

- ◆ **GetAttachOffsetFeet()**

```
virtual HRESULT GetAttachOffsetFeet( _out P3D::DXYZ & vOffset ) const [private] [virtual]
```

Gets the offset on the pylon in which it is attached to the parent
- ◆ **GetOwnerId()**

```
virtual uint GetOwnerId( ) const [private] [virtual]
```

Gets the ID of the object that the pylon is attached
- ◆ **SetOwnerId()**

```
virtual HRESULT SetOwnerId( _in uint uOwnerId ) [private] [virtual]
```

Sets the ID of the object that the pylon is attached

◆ P3D::ArticulatedPart

class P3D::ArticulatedPart

Class Members	
float	m_fPadding
float	m_fParameterValue
unsigned int	m_uiParameterType
unsigned short	m_usAttachedTold
unsigned char	m_yChangeIndicator
unsigned char	m_yRecordType

◆ P3D::ArticulatedParameter

class P3D::ArticulatedParameter

Class Members	
union ArticulatedParameter	_unnamed_

◆ P3D::IPduBuilderV440

class P3D::IPduBuilderV440

This interface allows developers to build PDU's on a per-field basis.

Remarks

It is the developer's responsibility to fill in the PDU header as well as any necessary data.

Inherits IUnknown.

Private Member Functions

```

virtual void WriteChar( _in char c ) PURE
virtual void WriteUChar( _in unsigned char uc ) PURE
virtual void WriteFloat( _in float f ) PURE
virtual void WriteDouble( _in double d ) PURE
virtual void WriteInt( _in int i ) PURE
virtual void WriteUInt( _in unsigned int u ) PURE
virtual void WriteLong( _in long l ) PURE
virtual void WriteULong( _in unsigned long ul ) PURE
virtual void WriteLongLong( _in long long ll ) PURE
virtual void WriteUShort( _in unsigned short us ) PURE
virtual void WriteShort( _in short s ) PURE
virtual int GetSize() const PURE

```

Member Function Documentation

- ◆ **GetSize()**
virtual int GetSize() const [private] [virtual]
Returns the current size of the PDU in bytes.
- ◆ **WriteChar()**
virtual void WriteChar (__in char c) [private] [virtual]
Adds an 8-bit signed byte to the packet.
- ◆ **WriteDouble()**
virtual void WriteDouble (__in double d) [private] [virtual]
Adds a 64-bit floating point to the packet.
- ◆ **WriteFloat()**
virtual void WriteFloat (__in float f) [private] [virtual]
Adds a 32-bit floating point to the packet.
- ◆ **WriteInt()**
virtual void WriteInt (__in int i) [private] [virtual]
Adds a 32-bit signed integer to the packet.
- ◆ **WriteLong()**
virtual void WriteLong (__in long l) [private] [virtual]
Adds a 32-bit signed long to the packet.
- ◆ **WriteLongLong()**
virtual void WriteLongLong (__in long long ll) [private] [virtual]
Adds a 64-bit signed long to the packet.
- ◆ **WriteShort()**
virtual void WriteShort (__in short s) [private] [virtual]
Adds a 16-bit signed short to the packet.
- ◆ **WriteUChar()**
virtual void WriteUChar (__in unsigned char uc) [private] [virtual]
Adds an 8-bit unsigned short to the packet.
- ◆ **WriteUInt()**
virtual void WriteUInt (__in unsigned int u) [private] [virtual]
Adds a 32-bit unsigned integer to the packet.
- ◆ **WriteULong()**
virtual void WriteULong (__in unsigned long ul) [private] [virtual]
Adds a 32-bit unsigned long to the packet.
- ◆ **WriteUShort()**
virtual void WriteUShort (__in unsigned short us) [private] [virtual]

Adds a 16-bit unsigned short to the packet.

◆ P3D::IPduReaderV440

class P3D::IPduReaderV440

This interface allows developers to read PDU's on a per-field basis.

Inherits IUnknown.

Private Member Functions

```
virtual char ReadChar () PURE  
virtual unsigned char ReadUChar () PURE  
virtual float ReadFloat () PURE  
virtual double ReadDouble () PURE  
virtual int ReadInt () PURE  
virtual unsigned int ReadUInt () PURE  
virtual long ReadLong () PURE  
virtual unsigned long ReadULong () PURE  
virtual long long ReadLongLong () PURE  
virtual unsigned short ReadUShort () PURE  
virtual short ReadShort () PURE  
virtual const char * GetRawData () const PURE  
virtual UINT GetSize () const PURE
```

Member Function Documentation

◆ GetRawData()

virtual const char* GetRawData () const [private] [virtual]

Returns the current data inside of the PDU.

◆ GetSize()

virtual UINT GetSize () const [private] [virtual]

Returns the current size of the PDU in bytes.

◆ ReadChar()

virtual char ReadChar () [private] [virtual]

Reads an 8-bit signed byte to the packet.

◆ ReadDouble()

virtual double ReadDouble () [private] [virtual]

Reads a 64-bit floating point to the packet.

◆ ReadFloat()

virtual float ReadFloat () [private] [virtual]

Reads a 32-bit floating point to the packet.

◆ ReadInt()

virtual int ReadInt () [private] [virtual]

Reads a 32-bit signed integer to the packet.

◆ ReadLong()

virtual long ReadLong () [private] [virtual]

Reads a 32-bit signed long to the packet.

◆ **ReadLongLong()**

virtual long long ReadLongLong() private virtual

Reads a 64-bit signed long to the packet.

◆ **ReadShort()**

virtual short ReadShort() private virtual

Reads a 16-bit signed short to the packet.

◆ **ReadUChar()**

virtual unsigned char ReadUChar() private virtual

Reads an 8-bit unsigned short to the packet.

◆ **ReadUInt()**

virtual unsigned int ReadUInt() private virtual

Reads a 32-bit unsigned integer to the packet.

◆ **ReadULong()**

virtual unsigned long ReadULong() private virtual

Reads a 32-bit unsigned long to the packet.

◆ **ReadUShort()**

virtual unsigned short ReadUShort() private virtual

Reads a 16-bit unsigned short to the packet.

◆ **P3D::IPduCallbackV440**

class P3D::IPduCallbackV440

This interface allows developers to create PDU's to be sent or received.

Inherits IUnknown.

Private Member Functions

virtual HRESULT **OnSend** (__in IPduReaderV440 *pReader, __in BYTE uPduType) **PURE**

virtual HRESULT **OnReceive** (__in IPduReaderV440 *pReader, __in BYTE uPduType)
PURE

Member Function Documentation

◆ **OnReceive()**

virtual HRESULT OnReceive (__in IPduReaderV440 * pReader,
 __in BYTE uPduType
)private virtual

Plugins should implement this function to receive callbacks when Prepar3D has received a packet.

Remarks

Returning anything other than S_OK will prevent the packet from being processed within Prepar3D.

◆ **OnSend()**

virtual HRESULT OnSend (__in IPduReaderV440 * pReader,
 __in BYTE uPduType
)private virtual

Plugins should implement this function to receive callbacks when Prepar3D is about to send a packet.

Remarks

Returning anything other than S_OK will prevent the packet from being sent over the network.

◆ P3D::IDISManagerV450

class P3D::IDISManagerV450

Professional Plus Only

This service allows the developer to interact and retrieve information with a distributed interactive simulation (DIS) session. Developers integrating with this interface should be familiar with and are expected to follow DIS IEEE standards. This service is provided by the **IPdk** interface.

Inherits IDISManagerV440.

Private Member Functions

virtual BOOL	IsConnected () const PURE
virtual HRESULT	GetEntityTypeById (_in UINT32 uID, __out P3D::EntityType & EntityType) const PURE
virtual HRESULT	NotifyMunitionFired (_in UINT32 uAttackerID, __in UINT32 uTargetID, __in UINT32 uMunitionID, __in const P3D::EntityType & EntityType , __in const P3D::DXYZ &xyzLonAltLat, __in const P3D::DXYZ &xyzLinearVelocity, __in unsigned short usWarheadType, __in unsigned short usFuseType, __in unsigned short usQuantity, __in unsigned short usRate, __in float fRange, __inout unsigned short &usEventID) PURE
virtual HRESULT	NotifyMunitionDetonated (_in UINT32 uAttackerID, __in UINT32 uTargetID, __in UINT32 uMunitionID, __in const P3D::EntityType & EntityType , __in unsigned short usEventID, __in const P3D::DXYZ &xyzLonAltLat, __in const P3D::DXYZ &xyzLinearVelocity, __in unsigned short usWarheadType, __in unsigned short usFuseType, __in unsigned short usQuantity, __in unsigned short usRate, __in unsigned char yDetonationResult) PURE
virtual HRESULT	GetEntityIdByObjectId (_in UINT32 uObjectId, __out unsigned short &usSitelid, __out unsigned short &usApplicationId, __out unsigned short &usEntityId) PURE
virtual HRESULT	GetObjectidByEntityId (_in unsigned short usSitelid, __in unsigned short usApplicationId, __in unsigned short usEntityId, __out UINT32 &uObjectId) PURE
virtual P3D::IPduBuilderV440 *	CreatePdu () PURE
virtual HRESULT	IssuePdu (P3D::IPduBuilderV440 *pPduBuilder) PURE
virtual HRESULT	RegisterPduCallback (_in BYTE yPduType, __in __notnull IPduCallbackV440 *pCallback) PURE
virtual HRESULT	UnregisterPduCallback (_in BYTE yPduType, __in __notnull IPduCallbackV440 *pCallback) PURE
virtual HRESULT	SetDisableReceive (_in BOOL bDisableReceive) PURE
virtual HRESULT	SetDisableSend (_in BOOL bDisableSend) PURE
virtual USHORT	GetSitelid () const PURE
virtual USHORT	GetApplicationId () const PURE
virtual BYTE	GetExerciseId () const PURE
virtual USHORT	GetEventId () const PURE
virtual int	GetWallClockHour () const PURE
virtual uint	GetWallTimestamp () const PURE
virtual int	GetSimClockHour () const PURE
virtual uint	GetSimTimestamp () const PURE

Member Function Documentation

◆ CreatePdu()

virtual P3D::IPduBuilderV440* CreatePdu () **private** **virtual**

Returns an **IPduBuilderV440** interface with a reference count of 1. This interface can be used to build PDU's to be used with the IssuePdu function. Developers should release this object after it has been issued using the IssuePdu function.

Sample implementation:

```
P3D::IPduBuilderV440* pPdu = spDIS->CreatePdu();  
  
// Write PDU header  
pPdu->WriteUChar(6);  
pPdu->WriteUChar(spDIS->GetExerciseId());
```

```

pPdu->WriteUChar(1);
pPdu->WriteUChar(1);
pPdu->WriteUInt(spDIS->GetSimTimestamp());
// ...

// Write remaining PDU specific data
// ...

spDIS->IssuePdu(pPdu);

pPdu->Release();
pPdu = nullptr;

```

Remarks

It is the developer's responsibility to fill in the PDU header as well as any necessary data.

◆ GetApplicationId()

virtual USHORT GetApplicationId() const private virtual

Returns the session's current application id.

◆ GetEntityIdByObjectId()

virtual HRESULT GetEntityIdByObjectId(__in UINT32 uObjectId,
__out unsigned short & usSiteId,
__out unsigned short & usApplicationId,
__out unsigned short & usEntityId
)

private virtual

Returns the entity identifier for the given object id if successful.

Parameters

uObjectId The object id of the request
usSiteId The site id of the entity identifier
usApplicationId The application id of the entity identifier
usEntityId The entity/object id of the entity identifier

◆ GetEntityTypeById()

virtual HRESULT GetEntityTypeById(__in UINT32 uID,
__out P3D::EntityType & EntityType
)

const private virtual

Provides the **EntityType** for the given object ID if successful.

◆ GetEventId()

virtual USHORT GetEventId() private virtual

Creates and returns a unique event id for the session. This value should be used when creating PDU's with the **IPduBuilderV440** interface that require an event ID.

◆ GetExerciseId()

virtual BYTE GetExerciseId() const private virtual

Returns the session's current exercise id.

◆ GetObjectIdByEntityId()

virtual HRESULT GetObjectIdByEntityId(__in unsigned short usSiteId,
__in unsigned short usApplicationId,
__in unsigned short usEntityId,
__out UINT32 & uObjectId
)

private virtual

Returns the object id for the given entity identifier if successful.

Parameters

usSiteId The site id of the entity identifier
usApplicationId The application id of the entity identifier
usEntityId The entity/object id of the entity identifier
uObjectId The object id of the request

◆ GetSimClockHour()

virtual int GetSimClockHour() const [private] [virtual]

Returns the session's current simulation clock hour since 0000 hours January 1, 1970 UTC.

◆ **GetSimTimestamp()**

virtual UINT GetSimTimestamp() const [private] [virtual]

Returns the session's current simulation timestamp in DIS timestamp format.

◆ **GetSiteId()**

virtual USHORT GetSiteId() const [private] [virtual]

Returns the session's current site id.

◆ **GetWallClockHour()**

virtual int GetWallClockHour() const [private] [virtual]

Returns the session's current wall clock hour since 0000 hours January 1, 1970 UTC.

◆ **GetWallTimestamp()**

virtual UINT GetWallTimestamp() const [private] [virtual]

Returns the session's current wall timestamp in DIS timestamp format. This value should be used when filling out the PDU header using the [IPduBuilderV440](#) interface.

◆ **IsConnected()**

virtual BOOL IsConnected() const [private] [virtual]

Returns TRUE if a DIS connection is active, FALSE otherwise.

◆ **IssuePdu()**

virtual HRESULT IssuePdu([P3D::IPduBuilderV440](#) * pPduBuilder) [private] [virtual]

Informs core [P3D](#) to queue the given [IPduBuilderV440](#) interface data to be broadcast across the network.

Remarks

The [IPduBuilderV440](#) object can be created with a call to CreatePdu.

This function does not add a ref to the given [IPduBuilderV440](#) interface.

◆ **NotifyMunitionDetonated()**

virtual HRESULT NotifyMunitionDetonated(__in [UINT32](#) uAttackerID,
__in [UINT32](#) uTargetID,
__in [UINT32](#) uMunitionID,
__in const [P3D::EntityType](#) & EntityType,
__in unsigned short usEventID,
__in const [P3D::DXYZ](#) & xyzLonAltLat,
__in const [P3D::DXYZ](#) & xyzLinearVelocity,
__in unsigned short usWarheadType,
__in unsigned short usFuseType,
__in unsigned short usQuantity,
__in unsigned short usRate,
__in unsigned char yDetonationResult
)

[private] [virtual]

Used to issue a Detonation PDU.

Parameters

uAttackerID	The object ID of the firing entity
uTargetID	The object ID of the target entity if available, 0 otherwise
uMunitionID	The object ID of the munition entity if available, 0 otherwise
EntityType	The EntityType of the munition
usEventID	The event ID from an associated Fire PDU if available, 0 otherwise
xyzLonAltLat	Radians/feet
xyzLinearVelocity	World/FPS
usWarheadType	The warhead type

usFuseType The fuse type
usQuantity The quantity of munitions represented
usRate The rate of fire in rounds per minute
yDetonationResult The result of the detonation

◆ NotifyMunitionFired()

```
virtual HRESULT
NotifyMunitionFired( ( __in UINT32 uAttackerID,
                      __in UINT32 uTargetID,
                      __in UINT32 uMunitionID,
                      __in const P3D::EntityType & EntityType,
                      __in const P3D::DXYZ & xyzLonAltLat,
                      __in const P3D::DXYZ & xyzLinearVelocity,
                      __in unsigned short usWarheadType,
                      __in unsigned short usFuseType,
                      __in unsigned short usQuantity,
                      __in unsigned short usRate,
                      __in float fRange,
                      __inout unsigned short & usEventID
                    )
) private virtual
```

Used to issue a Fire PDU.

Parameters

uAttackerID The object ID of the firing entity
uTargetID The object ID of the target entity if available, 0 otherwise
uMunitionID The object ID of the munition entity if available, 0 otherwise
EntityType The **EntityType** of the munition
xyzLonAltLat World location in radians and feet
xyzLinearVelocity World velocity in feet per second
usWarheadType The warhead type
usFuseType The fuse type
usQuantity The quantity of munitions represented
usRate Rounds per minute
fRange Meters
usEventID Set to 0 for new Fire PDU or previously returned value to signify continuous firing

◆ RegisterPduCallback()

```
virtual HRESULT
RegisterPduCallback( ( __in BYTE yPduType,
                      __in __notnull IPduCallbackV440 * pCallback
                    )
) private virtual
```

Registers a PDU callback.

◆ SetDisableReceive()

```
virtual HRESULT SetDisableReceive ( __in BOOL bDisableReceive ) private virtual
```

Plugins can toggle whether Prepar3D receives any packets.

Parameters

bDisableReceive True to disable receiving, False to enable receiving.

Remarks

By default receiving is enabled.

◆ SetDisableSend()

```
virtual HRESULT SetDisableSend ( __in BOOL bDisableSend ) private virtual
```

Plugins can toggle whether Prepar3D sends any packets.

Parameters

bDisableSend True to disable sending, False to enable sending.

Remarks

By default sending is enabled.

◆ UnregisterPduCallback()

```
virtual HRESULT
UnregisterPduCallback( ( __in BYTE yPduType,
                        __in __notnull IPduCallbackV440 * pCallback
                      )
)
```

private virtual

Unregisters a PDU callback.

◆ P3D::IDISServiceV400

Professional Plus Only

This service allows developers to provide Distributed Interactive Simulation (DIS) information to the core simulation. Developers should implement this service and provide the requested information following DIS IEEE standards.

Inherits IUnknown.

Private Member Functions

```
virtual HRESULT SerializeEntityAppearance (__inout UINT &iAppearance) PURE
virtual HRESULT DeserializeEntityAppearance (__in UINT iAppearance) PURE
virtual HRESULT GetArticulatedParameterCount (__inout UINT &iCount) PURE
virtual HRESULT SerializeArticulatedParameter (__in UINT iIndex, __inout ArticulatedParameter &ArticulatedParam) PURE
virtual HRESULT DeserializeArticulatedParameter (__in UINT iIndex, __in const ArticulatedParameter &ArticulatedParam) PURE
```

Member Function Documentation

◆ **DeserializeArticulatedParameter()**

```
virtual HRESULT
DeserializeArticulatedParameter ( __in UINT iIndex,
__in const ArticulatedParameter & ArticulatedParam
)
```

private virtual

This function is called on remote objects when the given articulated parameter needs to be updated. This function maybe called by the application when an entity state PDU is received or due to the dead reckoning of the parameter. The function should return S_OK if the articulated parameter was correctly serialized.

◆ **DeserializeEntityAppearance()**

```
virtual HRESULT DeserializeEntityAppearance ( __in UINT iAppearance ) private virtual
```

This function will be called on remote entities when the appearance needs to be updated. The data should be deserialized in the same manner as described above. The function should return S_OK if the entity appearance was correctly deserialized.

◆ **GetArticulatedParameterCount()**

```
virtual HRESULT GetArticulatedParameterCount ( __inout UINT & iCount ) private virtual
```

This function is called when requesting the articulated parameter count. The function should return S_OK and provide the number of articulated parameters if the ISimObject is providing articulated parameter support.

◆ **SerializeArticulatedParameter()**

```
virtual HRESULT
SerializeArticulatedParameter ( __in UINT iIndex,
__inout ArticulatedParameter & ArticulatedParam
)
```

private virtual

This function is called on a given articulated parameter when the application is requesting an update. The **ArticulatedParameter** union class should be filled out in accordance to DIS standards. This function maybe called by the application when an entity state PDU is required due to heartbeat duration or articulated parameter position or rotation threshold values being exceeded. The function should return S_OK if the articulated parameter was correctly serialized.

◆ **SerializeEntityAppearance()**

```
virtual HRESULT SerializeEntityAppearance ( __inout UINT & iAppearance ) private virtual
```

This function will be called when the application requires the object's entity appearance. The entity appearance is a 32-bit unsigned integer. The application expects the data to be packed according to DIS standards. The application expects the appearance type to match that of the entity type and domain (platform, air, land, munition, expendable, etc.). The function should return S_OK if the entity appearance is being provided by the ISimObject.

◆ P3D::ArticulatedParameter.__unnamed__

union P3D::ArticulatedParameter.__unnamed__

Class Members

ArticulatedPart m_ArticulatedPart

Variables

GUID IID_IAIBehaviorManagerV01

GUID SID_AIBehaviorManager

GUID SID_AIBehavior

GUID IID_IAIBehaviorWingmanFormationV01

GUID SID_AIBehaviorWingmanFormation

GUID IID_IAIBehaviorAttackerV400

GUID SID_AIBehaviorAttacker

GUID IID_IAIBehaviorPursueV500

GUID SID_AIBehaviorPursue

GUID IID_IAIBehaviorCombatAirPatrolV01

GUID SID_AIBehaviorCombatAirPatrol

GUID IID_IAIBehaviorCloseAirSupportV01

GUID SID_AIBehaviorCloseAirSupport

GUID IID_IAIBehaviorSearchTrackV01

GUID SID_AIBehaviorSearchTrack

GUID IID_ISimObjectAIV02

GUID SID_SimObjectAI

GUID SID_AIService

GUID SID_AircraftAIService

GUID IID_IAirplaneAIServiceV02

GUID SID_AirplaneAIService

GUID IID_IHelicopterAIServiceV420

GUID SID_HelicopterAIService

GUID IID_IGroundVehicleAIServiceV01

GUID SID_GroundVehicleAIService

GUID IID_IWaponsSystemV440

GUID SID_WaponsSystem

GUID IID_IWeaponServiceV420

GUID SID_WeaponService

GUID IID_ICountermeasureSystemV01

GUID SID_CountermeasureSystem

GUID IID_ICountermeasureServiceV02

GUID SID_CountermeasureService

GUID IID_IGunSystemV440

GUID SID_GunSystem

GUID IID_IGunV400

GUID SID_Gun

GUID IID_IFireControlSystemV01

GUID SID_FireControlSystem

GUID IID_IGuidanceSystemV01

GUID SID_GuidanceSystem

GUID IID_IPylonServiceV01

GUID SID_PylonService

GUID IID_IPduBuilderV440

GUID IID_IPduReaderV440

GUID IID_IPduCallbackV440

GUID IID_IDISManagerV450

GUID SID_DISManager

GUID IID_IDISServiceV400

GUID SID_DISService

Variable Documentation

◆ IID_IAIBehaviorAttackerV400

GUID IID_IAIBehaviorAttackerV400

- ◆ IID_IAIBehaviorCloseAirSupportV01
GUID IID_IAIBehaviorCloseAirSupportV01
- ◆ IID_IAIBehaviorCombatAirPatrolV01
GUID IID_IAIBehaviorCombatAirPatrolV01
- ◆ IID_IAIBehaviorManagerV01
GUID IID_IAIBehaviorManagerV01
- ◆ IID_IAIBehaviorPursueV500
GUID IID_IAIBehaviorPursueV500
- ◆ IID_IAIBehaviorSearchTrackV01
GUID IID_IAIBehaviorSearchTrackV01
- ◆ IID_IAIBehaviorWingmanFormationV01
GUID IID_IAIBehaviorWingmanFormationV01
- ◆ IID_IAirplaneAIServiceV02
GUID IID_IAirplaneAIServiceV02
- ◆ IID_ICountermeasureServiceV02
GUID IID_ICountermeasureServiceV02
- ◆ IID_ICountermeasureSystemV01
GUID IID_ICountermeasureSystemV01
- ◆ IID_IDISManagerV450
GUID IID_IDISManagerV450
- ◆ IID_IDISServiceV400
GUID IID_IDISServiceV400
- ◆ IID_IFireControlSystemV01
GUID IID_IFireControlSystemV01
- ◆ IID_IGroundVehicleAIServiceV01
GUID IID_IGroundVehicleAIServiceV01
- ◆ IID_IGuidanceSystemV01
GUID IID_IGuidanceSystemV01
- ◆ IID_IGunSystemV440
GUID IID_IGunSystemV440
- ◆ IID_IGunV400
GUID IID_IGunV400
- ◆ IID_IHelicopterAIServiceV420
GUID IID_IHelicopterAIServiceV420

- ◆ IID_IPduBuilderV440
GUID IID_IPduBuilderV440
- ◆ IID_IPduCallbackV440
GUID IID_IPduCallbackV440
- ◆ IID_IPduReaderV440
GUID IID_IPduReaderV440
- ◆ IID_IPylonServiceV01
GUID IID_IPylonServiceV01
- ◆ IID_ISimObjectAIv02
GUID IID_ISimObjectAIv02
- ◆ IID_IWeaponServiceV420
GUID IID_IWeaponServiceV420
- ◆ IID_IWeaponsSystemV440
GUID IID_IWeaponsSystemV440
- ◆ SID_AIBehavior
GUID SID_AIBehavior
- ◆ SID_AIBehaviorAttacker
GUID SID_AIBehaviorAttacker
- ◆ SID_AIBehaviorCloseAirSupport
GUID SID_AIBehaviorCloseAirSupport
- ◆ SID_AIBehaviorCombatAirPatrol
GUID SID_AIBehaviorCombatAirPatrol
- ◆ SID_AIBehaviorManager
GUID SID_AIBehaviorManager
- ◆ SID_AIBehaviorPursue
GUID SID_AIBehaviorPursue
- ◆ SID_AIBehaviorSearchTrack
GUID SID_AIBehaviorSearchTrack
- ◆ SID_AIBehaviorWingmanFormation
GUID SID_AIBehaviorWingmanFormation
- ◆ SID_AircraftAIService
GUID SID_AircraftAIService
- ◆ SID_AirplaneAIService
GUID SID_AirplaneAIService

GUID SID_AirplaneAIService
◆ SID_AIService
GUID SID_AIService
◆ SID_CountermeasureService
GUID SID_CountermeasureService
◆ SID_CountermeasureSystem
GUID SID_CountermeasureSystem
◆ SID_DISManager
GUID SID_DISManager
◆ SID_DISService
GUID SID_DISService
◆ SID_FireControlSystem
GUID SID_FireControlSystem
◆ SID_GroundVehicleAIService
GUID SID_GroundVehicleAIService
◆ SID_GuidanceSystem
GUID SID_GuidanceSystem
◆ SID_Gun
GUID SID_Gun
◆ SID_GunSystem
GUID SID_GunSystem
◆ SID_HelicopterAIService
GUID SID_HelicopterAIService
◆ SID_PylonService
GUID SID_PylonService
◆ SID_SimObjectAI
GUID SID_SimObjectAI
◆ SID_WeaponService
GUID SID_WeaponService
◆ SID_WeaponsSystem
GUID SID_WeaponsSystem

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Rendering Services

Overview

The rendering services enables external applications to render custom content into a Prepar3D 3D view. The rendering plugin system allows for custom DirectX 12 rendering into textures or overlay onto a view using rendering plug-ins. The object rendering services provides a way to add custom objects such as lights to a 3d scene.

Classes

- struct [PdkRenderFlags](#)
- struct [TextureDescriptionV440](#)
- class [IRenderingPluginSystemV400](#)
- class [IRenderingPluginSystemV430](#)
- class [IRenderingPluginSystemV440](#)
- class [IRenderingPluginV400](#)
- class [IRenderDataV400](#)
- class [IRenderDataV430](#)
- struct [TextureDescriptionV500](#)
- class [IRenderingPluginSystemV500](#)
- class [IRenderingPluginV500](#)
- class [IRenderDataResourceV500](#)
- class [IRenderDataV500](#)
- class [IObjectRendererV500](#)
- class [IDynamicLightDataV500](#)
- class [RenderingPlugin](#)

Class Documentation

◆ P3D::PdkRenderFlags

struct P3D::PdkRenderFlags

This class holds a set of flags that describe how the render data will be used during the Render() callback. These should be set using SetRenderFlags during the PreRender() callback.

Remarks

The [IRenderDataV500](#) provided to the Render() call will only contain resources that were requested with these flags.

Public Member Functions

[PdkRenderFlags \(\) noexcept](#)

Public Attributes

bool [RenderingIsEnabled: 1](#)

```
bool WillWriteColor: 1  
bool WillWriteDepthStencil: 1  
bool WillReadColor: 1  
bool WillReadDepthStencil: 1  
bool WillModifyDeviceState: 1  
bool Output8bpp: 1
```

Constructor & Destructor Documentation

◆ PdkRenderFlags()

PdkRenderFlags () **inline** **noexcept**

Member Data Documentation

◆ Output8bpp

bool Output8bpp

Set This to ensure an 8888 format

◆ RenderingIsEnabled

bool RenderingIsEnabled

If false, the Render() function will not be called

◆ WillModifyDeviceState

bool WillModifyDeviceState

Set this to true if any device/content states will be set by the render call.

◆ WillReadColor

bool WillReadColor

If false, InputColor will not be provided

◆ WillReadDepthStencil

bool WillReadDepthStencil

If false, InputDepthStencil will not be provided

◆ WillWriteColor

bool WillWriteColor

If false, OutputColor will not be provided

◆ WillWriteDepthStencil

bool WillWriteDepthStencil

If false, OutputDepthStencil will not be provided

◆ P3D::TextureDescriptionV440

struct P3D::TextureDescriptionV440

Class Members

bool	bFrameDependent
DXGI_FORMAT	eFormat
IRenderingPluginV400 *	pPlugin
const WCHAR *	szName
unsigned int	uHeight
unsigned int	uWidth

◆ P3D::IRenderingPluginSystemV400

class P3D::IRenderingPluginSystemV400

Inherits IUnknown.

Inherited by **IRenderingPluginSystemV430**.

Private Member Functions

```
virtual HRESULT CreateTexture (const WCHAR *name, unsigned int width, unsigned int height, IRenderingPluginV400 *plugin)=0  
virtual HRESULT RemoveTexture (const WCHAR *name)=0  
virtual HRESULT GetCreatedTextures (INameList &nameList)=0  
virtual HRESULT CreateEffect (const WCHAR *name, IRenderingPluginV400 *plugin)=0  
virtual HRESULT RemoveEffect (const WCHAR *name)=0  
virtual HRESULT GetSystemEffects (INameList &names)=0
```

Member Function Documentation

◆ CreateEffect()

```
virtual HRESULT CreateEffect ( const WCHAR * name,  
                               IRenderingPluginV400 * plugin  
                           )  
                           [private] [pure virtual]
```

Create a new effect with given name and a callback function for each update

Parameters

name Name of Effect
plugin plugin that will be used to render the effect

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implemented in **IRenderingPluginSystemV440**, and **IRenderingPluginSystemV430**.

◆ CreateTexture()

```
virtual HRESULT CreateTexture ( const WCHAR * name,  
                               unsigned int width,  
                               unsigned int height,  
                               IRenderingPluginV400 * plugin  
                           )  
                           [private] [pure virtual]
```

Create a new texture given name, size, and a callback function for each update

Parameters

name Name of texture. Models and gauges can map to this texture by name.
width Width of texture in pixels
height Height of texture in pixels

plugin plugin that will be used to render into the texture

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

Implemented in **IRenderingPluginSystemV440**, and **IRenderingPluginSystemV430**.

◆ **GetCreatedTextures()**

virtual HRESULT GetCreatedTextures (**INameList** & **nameList**) **private** **pure virtual**

Get list of textures created externally

Parameters

[out] **nameList** list of names of all plugins

Implemented in **IRenderingPluginSystemV440**, and **IRenderingPluginSystemV430**.

◆ **GetSystemEffects()**

virtual HRESULT GetSystemEffects (**INameList** & **names**) **private** **pure virtual**

List of effects provided by the system

Parameters

[out] **names** list of names of effects provided by the system

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implemented in **IRenderingPluginSystemV440**, and **IRenderingPluginSystemV430**.

◆ **RemoveEffect()**

virtual HRESULT RemoveEffect (const WCHAR * **name**) **private** **pure virtual**

Remove an effect with given name that was created externally

Parameters

name Name of Effect

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implemented in **IRenderingPluginSystemV440**, and **IRenderingPluginSystemV430**.

◆ **RemoveTexture()**

virtual HRESULT RemoveTexture (const WCHAR * **name**) **private** **pure virtual**

Remove a texture given the texture name

Parameters

name Name of plugin

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implemented in **IRenderingPluginSystemV440**, and **IRenderingPluginSystemV430**.

◆ **P3D::IRenderingPluginSystemV430**

class P3D::IRenderingPluginSystemV430

Plugin service used to register custom rendering plugins. These plugins can be used to render into a texture, or to render on top of an existing view. If requested, a plugin can also read from the current view output allowing it to implement post process effects that are too complex to be

implemented through the xml/hlsl based custom post process system.

Inherits [IRenderingPluginSystemV400](#).

Inherited by [IRenderingPluginSystemV440](#).

Private Member Functions

```
virtual HRESULT CreateTexture (const WCHAR *name, unsigned int width, unsigned int height, IRenderingPluginV400 *plugin)=0  
virtual HRESULT RemoveTexture (const WCHAR *name)=0  
virtual HRESULT GetCreatedTextures (INameList &nameList)=0  
virtual HRESULT CreateEffect (const WCHAR *name, IRenderingPluginV400 *plugin)=0  
virtual HRESULT RemoveEffect (const WCHAR *name)=0  
virtual HRESULT GetSystemEffects (INameList &names)=0  
virtual HRESULT GetDeviceWindow (UINT uAdapterID, HWND &hWindow)=0
```

Member Function Documentation

◆ CreateEffect()

```
virtual HRESULT CreateEffect ( const WCHAR * name,  
                               IRenderingPluginV400 * plugin  
                           )  
private pure virtual
```

Create a new effect with given name and a callback function for each update

Parameters

name Name of Effect
plugin plugin that will be used to render the effect

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implements [IRenderingPluginSystemV400](#).

Implemented in [IRenderingPluginSystemV440](#).

◆ CreateTexture()

```
virtual HRESULT CreateTexture ( const WCHAR * name,  
                               unsigned int width,  
                               unsigned int height,  
                               IRenderingPluginV400 * plugin  
                           )  
private pure virtual
```

Create a new texture given name, size, and a callback function for each update

Parameters

name Name of texture. Models and gauges can map to this texture by name.
width Width of texture in pixels
height Height of texture in pixels
plugin plugin that will be used to render into the texture

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

Implements [IRenderingPluginSystemV400](#).

Implemented in [IRenderingPluginSystemV440](#).

◆ GetCreatedTextures()

```
virtual HRESULT GetCreatedTextures ( INameList & nameList )  
private pure virtual
```

Get list of textures created externally

Parameters[out] **nameList** list of names of all pluginsImplements **IRenderingPluginSystemV400**.Implemented in **IRenderingPluginSystemV440**.**◆ GetDeviceWindow()**

```
virtual HRESULT GetDeviceWindow ( UINT      uAdapterID,  
                                 HWND & hWindow  
                           )
```

private pure virtual

Get a Win32 HWND by adapter ID. This provides a window associated with a specific GPU using the AdapterID provided by IRenderData.

Parameters**uAdapterID** Adapter ID for a GPU[out] **hWindow** Reference to an HWND window handle**Returns**

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implemented in **IRenderingPluginSystemV440**.**◆ GetSystemEffects()**

```
virtual HRESULT GetSystemEffects ( INamelist & names )
```

private pure virtual

List of effects provided by the system

Parameters[out] **names** list of names of effects provided by the system**Returns**

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implemented in **IRenderingPluginSystemV400**.Implemented in **IRenderingPluginSystemV440**.**◆ RemoveEffect()**

```
virtual HRESULT RemoveEffect ( const WCHAR * name )
```

private pure virtual

Remove an effect with given name that was created externally

Parameters**name** Name of Effect**Returns**

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implemented in **IRenderingPluginSystemV400**.Implemented in **IRenderingPluginSystemV440**.**◆ RemoveTexture()**

```
virtual HRESULT RemoveTexture ( const WCHAR * name )
```

private pure virtual

Remove a texture given the texture name

Parameters**name** Name of plugin**Returns**

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implements [IRenderingPluginSystemV400](#).

Implemented in [IRenderingPluginSystemV440](#).

◆ P3D::IRenderingPluginSystemV440

class P3D::IRenderingPluginSystemV440

Plugin service used to register custom rendering plugins. These plugins can be used to render into a texture, or to render on top of an existing view. If requested, a plugin can also read from the current view output allowing it to implement post process effects that are too complex to be implemented through the xml/hlsl based custom post process system.

Inherits [IRenderingPluginSystemV430](#).

Private Member Functions

```
virtual HRESULT CreateTexture (const WCHAR *name, unsigned int width, unsigned int height, IRenderingPluginV400 *plugin)=0  
virtual HRESULT RemoveTexture (const WCHAR *name)=0  
virtual HRESULT GetCreatedTextures (INameList &nameList)=0  
virtual HRESULT CreateEffect (const WCHAR *name, IRenderingPluginV400 *plugin)=0  
virtual HRESULT RemoveEffect (const WCHAR *name)=0  
virtual HRESULT GetSystemEffects (INameList &names)=0  
virtual HRESULT GetDeviceWindow (UINT uAdapterID, HWND &hWindow)=0  
virtual HRESULT CreateTexture (const TextureDescriptionV440 &textureDesc)=0  
    virtual UINT GetAFRGroup ()=0  
    virtual bool IsMultiProjectionEnabled ()=0  
    virtual void SetMultiProjectionEnabled (bool bEnabled)=0
```

Member Function Documentation

◆ CreateEffect()

```
virtual HRESULT CreateEffect ( const WCHAR * name,  
                           IRenderingPluginV400 * plugin  
                           )  
                           [private] [pure virtual]
```

Create a new effect with given name and a callback function for each update

Parameters

name Name of Effect
plugin plugin that will be used to render the effect

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implements [IRenderingPluginSystemV430](#).

◆ CreateTexture() [1/2]

```
virtual HRESULT CreateTexture ( const WCHAR * name,  
                           unsigned int width,  
                           unsigned int height,  
                           IRenderingPluginV400 * plugin  
                           )  
                           [private] [pure virtual]
```

Create a new texture given name, size, and a callback function for each update

Parameters

name Name of texture. Models and gauges can map to this texture by name.
width Width of texture in pixels
height Height of texture in pixels
plugin plugin that will be used to render into the texture

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

Implements [IRenderingPluginSystemV430](#).

◆ CreateTexture() [2/2]

virtual HRESULT CreateTexture (const [TextureDescriptionV440](#) & textureDesc) [private](#) [pure virtual](#)

Create a new texture given name, size, and a callback function for each update

Parameters

textureDesc description of the texture to be created

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ GetAFRGroup()

virtual UINT GetAFRGroup () [private](#) [pure virtual](#)

Get alternate frame rendering (AFR) group index. When using SLI, this value indicates which GPU is in use. Some rendering plugins may need this information to keep resources synchronized

Returns

UINT, index of current alternate frame rendering group

◆ GetCreatedTextures()

virtual HRESULT GetCreatedTextures ([INameList](#) & nameList) [private](#) [pure virtual](#)

Get list of textures created externally

Parameters

[out] **nameList** list of names of all plugins

Implements [IRenderingPluginSystemV430](#).

◆ GetDeviceWindow()

virtual HRESULT GetDeviceWindow (UINT uAdapterID,
HWND & hWindow
) [private](#) [pure virtual](#)

Get a Win32 HWND by adapter ID. This provides a window associated with a specific GPU using the AdapterID provided by IRenderData.

Parameters

uAdapterID Adapter ID for a GPU

[out] **hWindow** Reference to an HWND window handle

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implements [IRenderingPluginSystemV430](#).

◆ GetSystemEffects()

virtual HRESULT GetSystemEffects ([INameList](#) & names) [private](#) [pure virtual](#)

List of effects provided by the system

Parameters

[out] **names** list of names of effects provided by the system

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implements [IRenderingPluginSystemV430](#).

◆ **IsMultiProjectionEnabled()**

virtual bool IsMultiProjectionEnabled() private pure virtual

Is multi-project enabled. This is used for SinglePass VR

Returns

bool, true if multi-projection is enabled

◆ **RemoveEffect()**

virtual HRESULT RemoveEffect (const WCHAR * **name**) private pure virtual

Remove an effect with given name that was created externally

Parameters

name Name of Effect

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implements [IRenderingPluginSystemV430](#).

◆ **RemoveTexture()**

virtual HRESULT RemoveTexture (const WCHAR * **name**) private pure virtual

Remove a texture given the texture name

Parameters

name Name of plugin

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

Implements [IRenderingPluginSystemV430](#).

◆ **SetMultiProjectionEnabled()**

virtual void SetMultiProjectionEnabled (bool **bEnabled**) private pure virtual

Set multi-project enabled. Do this to enable Single-Pass for custom VR plugin creation

Returns

bool, true if multi-projection is enabled

◆ **P3D::IRenderingPluginV400**

class P3D::IRenderingPluginV400

Rendering plugin interface used to implement texture and effect plugins. These plugins can be used to render into a texture, or to render on top of an existing view. If requested, a plugin can also read from the current view output allowing it to implement post process effects that are too complex to be implemented through the xml/hlsl based custom post process system.

Inherits [IUnknown](#).

Private Member Functions

virtual void [Render \(IRenderDataV400 *nRenderData\)](#) override

```
virtual void PreRender (IRenderDataV400 *pRenderData) override
```

Member Function Documentation

◆ PreRender()

```
virtual void PreRender (IRenderDataV400 * pRenderData) [private] [pure virtual]
```

Called before calling Render. This function should call pRenderData->SetRenderFlags() to let the plugin system know what resources in the render data will be read or written.

◆ Render()

```
virtual void Render (IRenderDataV400 * pRenderData) [private] [pure virtual]
```

A callback to render content

Parameters

pRenderData Interface to rendering device and resources used for rendering.

Remarks

Render will be called during each rendering pass unless the RenderingIsEnabled flag is set to false in PreRender.

◆ P3D::IRenderDataV400

class P3D::IRenderDataV400

Inherits IUnknown.

Inherited by **IRenderDataV430**.

Private Member Functions

```
virtual FLOAT GetTextureWidth ()=0  
virtual FLOAT GetTextureHeight ()=0  
virtual UINT GetAdapterID ()=0  
virtual ID3D11Device * GetDevice () override  
virtual ID3D11RenderTargetView * GetOutputColor () override  
virtual ID3D11DepthStencilView * GetOutputDepthStencil () override  
virtual ID3D11ShaderResourceView * GetInputColor () override  
virtual ID3D11ShaderResourceView * GetInputDepthStencil () override  
virtual ID3D11ShaderResourceView * GetTexture (const char *name) override  
virtual IWindowV400 * GetWindow () override  
virtual ICameraSystemV400 * GetCamera () override  
virtual PdkRenderFlags GetRenderFlags () override  
virtual void SetRenderFlags (PdkRenderFlags flags) override  
virtual PdkRenderPass GetRenderPass () override
```

Member Function Documentation

◆ GetAdapterID()

```
virtual UINT GetAdapterID () [private] [pure virtual]
```

Get Adapter id

Returns

Current Adapter of RenderData

Implemented in **IRenderDataV430**.

◆ **GetCamera()**

virtual ICameraSystemV400* GetCamera() **private** **pure virtual**

Get Camera

Returns

 Camera used for rendering

Implemented in **IRenderDataV430**.

◆ **GetDevice()**

virtual ID3D11Device* GetDevice() **private** **pure virtual**

Get Device

Returns

 D3D11 device.

Implemented in **IRenderDataV430**.

◆ **GetInputColor()**

virtual ID3D11ShaderResourceView* GetInputColor() **private** **pure virtual**

Get input color

Returns

 input color shader resource view.

Remarks

 Must set WillReadColor render flag to true in PreRender to use this

Implemented in **IRenderDataV430**.

◆ **GetInputDepthStencil()**

virtual ID3D11ShaderResourceView* GetInputDepthStencil() **private** **pure virtual**

Get Input Depth Stencil

Returns

 Input depth stencil as shader resource view.

Remarks

 Must set WillReadDepthStencil render flag to true in PreRender to use this

Implemented in **IRenderDataV430**.

◆ **GetOutputColor()**

virtual ID3D11RenderTargetView* GetOutputColor() **private** **pure virtual**

Get Output Color

Returns

 Output color render target view.

Remarks

 Must set WillWriteColor render flag to true in PreRender to use this

Implemented in **IRenderDataV430**.

◆ **GetOutputDepthStencil()**

virtual ID3D11DepthStencilView* GetOutputDepthStencil() [private] [pure virtual]

Get Output Depth Stencil

Returns

Output depth stencil view.

Remarks

Must set WillWriteDepthStencil render flag to true in PreRender to use this

Implemented in [IRenderDataV430](#).

◆ [GetRenderFlags\(\)](#)

virtual PdkRenderFlags GetRenderFlags() [private] [pure virtual]

Get PdkRenderFlags

Returns

PdkRenderFlags

Implemented in [IRenderDataV430](#).

◆ [GetRenderPass\(\)](#)

virtual PdkRenderPass GetRenderPass() [private] [pure virtual]

Get Render Pass

Returns

Current render pass.

Implemented in [IRenderDataV430](#).

◆ [GetTexture\(\)](#)

virtual ID3D11ShaderResourceView* GetTexture(const char* name) [private] [pure virtual]

Get texture by name

Parameters

name name of texture

Returns

shader resource view of texture

Implemented in [IRenderDataV430](#).

◆ [GetTextureHeight\(\)](#)

virtual FLOAT GetTextureHeight() [private] [pure virtual]

Get Texture height

Returns

Texture height in pixels

Implemented in [IRenderDataV430](#).

◆ [GetTextureWidth\(\)](#)

virtual FLOAT GetTextureWidth() [private] [pure virtual]

Get Texture width

Returns

Texture width in pixels

Implemented in IRenderDataV430.

◆ **GetWindow()**

virtual IWindowV400* GetWindow() **private** **pure virtual**

Get Window

Returns

Window used for rendering

Implemented in IRenderDataV430.

◆ **SetRenderFlags()**

virtual void SetRenderFlags(**PdkRenderFlags** flags) **private** **pure virtual**

Set **PdkRenderFlags**. This must be called from PreRender() for Render() to work properly.

Parameters

flags render flags associated with the current window and render pass

Implemented in IRenderDataV430.

◆ **P3D::IRenderDataV430**

class P3D::IRenderDataV430

Inherits **IRenderDataV400**.

Private Member Functions

```
virtual FLOAT GetTextureWidth ()=0
virtual FLOAT GetTextureHeight ()=0
virtual UINT GetAdapterID ()=0
virtual ID3D11Device * GetDevice () override
virtual ID3D11RenderTargetView * GetOutputColor () override
virtual ID3D11DepthStencilView * GetOutputDepthStencil () override
virtual ID3D11ShaderResourceView * GetInputColor () override
virtual ID3D11ShaderResourceView * GetInputDepthStencil () override
virtual ID3D11ShaderResourceView * GetInputStencil () override
virtual ID3D11ShaderResourceView * GetTexture (const char *name) override
virtual IWindowV400 * GetWindow () override
virtual ICameraSystemV400 * GetCamera () override
virtual PdkRenderFlags GetRenderFlags () override
virtual void SetRenderFlags (PdkRenderFlags flags) override
virtual PdkRenderPass GetRenderPass () override
```

Member Function Documentation

◆ **GetAdapterID()**

virtual UINT GetAdapterID() **private** **pure virtual**

Get Adapter id

Returns

Current Adapter of RenderData

Implements **IRenderDataV400**.

◆ GetCamera()

virtual ICameraSystemV400* GetCamera() private pure virtual

Get Camera

Returns

Camera used for rendering

Implements [IRenderDataV400](#).

◆ GetDevice()

virtual ID3D11Device* GetDevice() private pure virtual

Get Device

Returns

D3D11 device.

Implements [IRenderDataV400](#).

◆ GetInputColor()

virtual ID3D11ShaderResourceView* GetInputColor() private pure virtual

Get input color

Returns

input color shader resource view.

Remarks

Must set WillReadColor render flag to true in PreRender to use this

Implements [IRenderDataV400](#).

◆ GetInputDepthStencil()

virtual ID3D11ShaderResourceView* GetInputDepthStencil() private pure virtual

Get Input Depth Stencil

Returns

Input depth stencil as shader resource view.

Remarks

Must set WillReadDepthStencil render flag to true in PreRender to use this

Implements [IRenderDataV400](#).

◆ GetInputStencil()

virtual ID3D11ShaderResourceView* GetInputStencil() private pure virtual

Get Input Stencil

Returns

Input stencil as shader resource view.

Remarks

Will be provided only for RenderPassPreVc.

Stencil available through green channel as uint.

◆ GetOutputColor()

virtual ID3D11RenderTargetView* GetOutputColor() private pure virtual

Get Output Color

Returns

Ouput color render target view.

Remarks

Must set WillWriteColor render flag to true in PreRender to use this

Implements [IRenderDataV400](#).

◆ **GetOutputDepthStencil()**

virtual ID3D11DepthStencilView* GetOutputDepthStencil() private pure virtual

Get Output Depth Stencil

Returns

Ouput depth stencil view.

Remarks

Must set WillWriteDepthStencil render flag to true in PreRender to use this

Implements [IRenderDataV400](#).

◆ **GetRenderFlags()**

virtual PdkRenderFlags GetRenderFlags() private pure virtual

Get PdkRenderFlags

Returns

PdkRenderFlags

Implements [IRenderDataV400](#).

◆ **GetRenderPass()**

virtual PdkRenderPass GetRenderPass() private pure virtual

Get Render Pass

Returns

Current render pass.

Implements [IRenderDataV400](#).

◆ **GetTexture()**

virtual ID3D11ShaderResourceView* GetTexture(const char * **name**) private pure virtual

Get texture by name

Parameters

name name of texture

Returns

shader resource view of texture

Implements [IRenderDataV400](#).

◆ **GetTextureHeight()**

virtual FLOAT GetTextureHeight() private pure virtual

Get Texture height

Returns

Texture height in pixels

Implements [IRenderDataV400](#).

◆ [GetTextureWidth\(\)](#)

virtual FLOAT GetTextureWidth() [private](#) [pure virtual](#)

Get Texture width

Returns

Texture width in pixels

Implements [IRenderDataV400](#).

◆ [GetWindow\(\)](#)

virtual IWindowV400* GetWindow() [private](#) [pure virtual](#)

Get Window

Returns

Window used for rendering

Implements [IRenderDataV400](#).

◆ [SetRenderFlags\(\)](#)

virtual void SetRenderFlags([PdkRenderFlags](#) flags) [private](#) [pure virtual](#)

Set [PdkRenderFlags](#). This must be called from PreRender() for Render() to work properly.

Parameters

flags render flags associated with the current window and render pass

Implements [IRenderDataV400](#).

◆ [P3D::TextureDescriptionV500](#)

struct P3D::TextureDescriptionV500

Class Members

bool	bFrameDependent
DXGI_FORMAT	eFormat
IRenderingPluginV500 *	pPlugin
const WCHAR *	szName
unsigned int	uHeight
unsigned int	uWidth

◆ [P3D::IRenderingPluginSystemV500](#)

class P3D::IRenderingPluginSystemV500

Plugin service used to register custom rendering plugins. These plugins can be used to render into a texture, or to render on top of an existing view. If requested, a plugin can also read from the current view output allowing it to implement post process effects that are too complex to be implemented through the xml/hlsl based custom post process system.

Inherits [IUnknown](#).

Private Member Functions

virtual HRESULT [CreateTexture](#)(const WCHAR *name, unsigned int width, unsigned int height, [IRenderingPluginV500](#) *plugin)=0

virtual HRESULT [RemoveTexture](#)(const WCHAR *name)=0

virtual HRESULT [GetCreatedTextures](#)(INamedList &namedList)=0

```
virtual HRESULT CreateEffect (const WCHAR *name, IRenderingPluginV500 *plugin)=0
virtual HRESULT RemoveEffect (const WCHAR *name)=0
virtual HRESULT GetSystemEffects (INameList &names)=0
virtual HRESULT GetDeviceWindow (UINT uAdapterID, HWND &hWindow)=0
virtual HRESULT CreateTexture (const TextureDescriptionV500 &textureDesc)=0
    virtual UINT GetAFRGroup ()=0
    virtual bool IsMultiProjectionEnabled ()=0
    virtual void SetMultiProjectionEnabled (bool bEnabled)=0
```

Member Function Documentation

◆ CreateEffect()

```
virtual HRESULT CreateEffect ( const WCHAR * name,
                               IRenderingPluginV500 * plugin
                           )
```

private pure virtual

Create a new effect with given name and a callback function for each update

Parameters

name Name of Effect
plugin plugin that will be used to render the effect

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

◆ CreateTexture() [1/2]

```
virtual HRESULT CreateTexture ( const WCHAR * name,
                               unsigned int width,
                               unsigned int height,
                               IRenderingPluginV500 * plugin
                           )
```

private pure virtual

Create a new texture given name, size, and a callback function for each update

Parameters

name Name of texture. Models and gauges can map to this texture by name.
width Width of texture in pixels
height Height of texture in pixels
plugin plugin that will be used to render into the texture

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ CreateTexture() [2/2]

```
virtual HRESULT CreateTexture ( const TextureDescriptionV500 & textureDesc )
```

private pure virtual

Create a new texture given name, size, and a callback function for each update

Parameters

textureDesc description of the texture to be created

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ GetAFRGroup()

```
virtual UINT GetAFRGroup ( )
```

private pure virtual

Get alternate frame rendering (AFR) group index. When using SLI, this value indicates which

GPU is in use. some rendering plugins may need this information to keep resources synchronized

Returns

UINT, index of current alternate frame rendering group

◆ GetCreatedTextures()

virtual HRESULT GetCreatedTextures (**INameList** & **nameList**) **private** **pure virtual**

Get list of textures created externally

Parameters

[out] **nameList** list of names of all plugins

◆ GetDeviceWindow()

virtual HRESULT GetDeviceWindow (**UINT** **uAdapterID**,
 HWND & **hWindow**
)

private **pure virtual**

Get a Win32 HWND by adapter ID. This provides a window associated with a specific GPU using the AdapterID provided by IRenderData.

Parameters

uAdapterID Adapter ID for a GPU

[out] **hWindow** Reference to an HWND window handle

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

◆ GetSystemEffects()

virtual HRESULT GetSystemEffects (**INameList** & **names**) **private** **pure virtual**

List of effects provided by the system

Parameters

[out] **names** list of names of effects provided by the system

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

◆ IsMultiProjectionEnabled()

virtual bool IsMultiProjectionEnabled () **private** **pure virtual**

Is multi-project enabled. This is used for SinglePass VR

Returns

bool, true if multi-projection is enabled

◆ RemoveEffect()

virtual HRESULT RemoveEffect (const WCHAR * **name**) **private** **pure virtual**

Remove an effect with given name that was created externally

Parameters

name Name of Effect

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

◆ RemoveTexture()

```
virtual HRESULT RemoveTexture ( const WCHAR * name ) [private] [pure virtual]
```

Remove a texture given the texture name

Parameters

name Name of plugin

Returns

HRESULT, ERR_NONE if function succeeds or ERR_FAIL if it fails

◆ SetMultiProjectionEnabled()

```
virtual void SetMultiProjectionEnabled ( bool bEnabled ) [private] [pure virtual]
```

Set multi-project enabled. Do this to enable Single-Pass for custom VR plugin creation

Returns

bool, true if multi-projection is enabled

◆ P3D::IRenderingPluginV500

```
class P3D::IRenderingPluginV500
```

Rendering plugin interface used to implement texture and effect plugins. These plugins can be used to render into a texture, or to render on top of an existing view. If requested, a plugin can also read from the current view output allowing it to implement post process effects that are too complex to be implemented through the xml/hlsl based custom post process system.

Inherits IUnknown.

Inherited by [RenderingPlugin](#).

Private Member Functions

```
virtual void Render (IRenderDataV500 *pRenderData) override
```

```
virtual void PreRender (IRenderDataV500 *pRenderData) override
```

Member Function Documentation

◆ PreRender()

```
virtual void PreRender ( IRenderDataV500 * pRenderData ) [private] [pure virtual]
```

Called before calling Render. This function should call pRenderData->SetRenderFlags() to let the plugin system know what resources in the render data will be read or written.

Implemented in [RenderingPlugin](#).

◆ Render()

```
virtual void Render ( IRenderDataV500 * pRenderData ) [private] [pure virtual]
```

A callback to render content

Parameters

pRenderData Interface to rendering device and resources used for rendering.

Remarks

Render will be called during each rendering pass unless the RenderingIsEnabled flag is set to false in PreRender.

Implemented in [RenderingPlugin](#).

◆ P3D::IRenderDataResourceV500

class P3D::IRenderDataResourceV500

Render Data Resource interface used to get the D3D12 Resource and track the current state.

Inherits IUnknown.

Private Member Functions

```
virtual ID3D12Resource * GetD3D12Resource () override  
virtual D3D12_RESOURCE_STATES GetResourceState () override  
virtual void SetResourceState (D3D12_RESOURCE_STATES eState) override
```

Member Function Documentation

◆ GetD3D12Resource()

virtual ID3D12Resource* GetD3D12Resource () private pure virtual

Get the D3D12 Resource

Returns

D3D12 Resource

◆ GetResourceState()

virtual D3D12_RESOURCE_STATES GetResourceState () private pure virtual

Get the resource state

Returns

current D3D12 Resource State

◆ SetResourceState()

virtual void SetResourceState (D3D12_RESOURCE_STATES eState) private pure virtual

Sets the resource state

Parameters

eState D3D12 Enumeration for desired resource state

Returns

shader resource view of texture

◆ P3D::IRenderDataV500

class P3D::IRenderDataV500

Inherits IUnknown.

Private Member Functions

```
virtual FLOAT GetTextureWidth ()=0  
virtual FLOAT GetTextureHeight ()=0  
virtual UINT GetAdapterID ()=0  
virtual ID3D12Device * GetDevice () override  
virtual ID3D12CommandQueue * GetCommandQueue  
    (D3D12_COMMAND_LIST_TYPE eType) override  
virtual IRenderDataResourceV500 * GetOutputColor () override  
virtual IRenderDataResourceV500 * GetOutputDepthStencil () override
```

```
virtual IRenderDataSourceV500 * GetOutputDepthStencil() override
virtual IRenderDataSourceV500 * GetInputColor() override
virtual IRenderDataSourceV500 * GetInputDepthStencil() override
virtual IRenderDataSourceV500 * GetInputStencil() override
virtual IRenderDataSourceV500 * GetTexture(const char *name) override
    virtual PdkRenderFlags GetRenderFlags() override
        virtual void SetRenderFlags(PdkRenderFlags flags) override
    virtual IWindowV400 * GetWindow() override
    virtual ICameraSystemV400 * GetCamera() override
    virtual PdkRenderPass GetRenderPass() override
```

Member Function Documentation

◆ GetAdapterID()

```
virtual UINT GetAdapterID() [private] [pure virtual]
```

Get Adapter id

Returns

Current Adapter of RenderData

◆ GetCamera()

```
virtual ICameraSystemV400* GetCamera() [private] [pure virtual]
```

Get Camera

Returns

Camera used for rendering

◆ GetCommandQueue()

```
virtual ID3D12CommandQueue* GetCommandQueue(D3D12_COMMAND_LIST_TYPE eType) [private] [pure virtual]
```

Get Command Queue

Parameters

eType D3D12 enumeration for desired command que type

Returns

D3D12 Command Queue.

◆ GetDevice()

```
virtual ID3D12Device* GetDevice() [private] [pure virtual]
```

Get Device

Returns

D3D12 device.

◆ GetInputColor()

```
virtual IRenderDataSourceV500* GetInputColor() [private] [pure virtual]
```

Get Input color

Returns

input color resource.

Remarks

Must set WillReadColor render flag to true in PreRender to use this

◆ GetInputDepthStencil()

virtual **IRenderDataSourceV500*** GetInputDepthStencil() **private** **pure virtual**

Get Input Depth Stencil

Returns

Input depth stencil resource.

Remarks

Must set WillReadDepthStencil render flag to true in PreRender to use this

◆ GetInputStencil()

virtual **IRenderDataSourceV500*** GetInputStencil() **private** **pure virtual**

Get Input Stencil

Returns

Input stencil resource.

Remarks

Will be provided only for RenderPassPreVc.

Stencil available through green channel as uint.

◆ GetOutputColor()

virtual **IRenderDataSourceV500*** GetOutputColor() **private** **pure virtual**

Get Output Color

Returns

Ouput color render target resource.

Remarks

Must set WillWriteColor render flag to true in PreRender to use this

◆ GetOutputDepthStencil()

virtual **IRenderDataSourceV500*** GetOutputDepthStencil() **private** **pure virtual**

Get Output Depth Stencil

Returns

Ouput depth stencil resource.

Remarks

Must set WillWriteDepthStencil render flag to true in PreRender to use this

◆ GetRenderFlags()

virtual **PdkRenderFlags** GetRenderFlags() **private** **pure virtual**

Get **PdkRenderFlags**

Returns

PdkRenderFlags

◆ GetRenderPass()

virtual **PdkRenderPass** GetRenderPass() **private** **pure virtual**

Get Render Pass

Returns

Current render pass.

◆ **GetTexture()**

virtual IRenderDataSourceV500* GetTexture (const char * **name**) **private pure virtual**

Get texture by name

Parameters

name name of texture

Returns

shader resource view of texture

◆ **GetTextureHeight()**

virtual FLOAT GetTextureHeight () **private pure virtual**

Get Texture height

Returns

Texture height in pixels

◆ **GetTextureWidth()**

virtual FLOAT GetTextureWidth () **private pure virtual**

Get Texture width

Returns

Texture width in pixels

◆ **GetWindow()**

virtual IWindowV400* GetWindow () **private pure virtual**

Get Window

Returns

Window used for rendering

◆ **SetRenderFlags()**

virtual void SetRenderFlags (**PdkRenderFlags flags**) **private pure virtual**

Set **PdkRenderFlags**. This must be called from PreRender() for Render() to work properly.

Parameters

flags render flags associated with the current window and render pass

◆ **IObjectRendererV500**

class IObjectRendererV500

Service for rendering into a view

Inherits IObjectRendererV440.

Private Member Functions

virtual HRESULT **DrawSphere** (const **ObjectWorldTransform** &location, float radius, **ARGBColor** color, **RenderFlags** renderFlags=0) override

virtual HRESULT **DrawCylinder** (const **ObjectWorldTransform** &location, float radius, float height,

```

        ARGBColor color, RenderFlags renderFlags=0) override
virtual HRESULT DrawLine (const LLADegreesMeters &startLocation, const
LLADegreesMeters &endLocation, float width, float height, ARGBColor color,
RenderFlags renderFlags=0) override
virtual HRESULT DrawRectangle (const ObjectWorldTransform &location, float width, float height,
float depth, ARGBColor color, RenderFlags renderFlags=0) override
virtual HRESULT DrawTriangle (const ObjectWorldTransform &location, float width, float height,
float depth, ARGBColor color, RenderFlags renderFlags=0) override
virtual HRESULT DrawText2D (int x, int y, LPCWSTR szText, ARGBColor textColor,
TextDescription &textDescription, RenderFlags renderFlags) override
virtual HRESULT DrawText3D (const ObjectWorldTransform &location, LPCWSTR szText,
ARGBColor textColor, TextDescription &textDescription, RenderFlags
renderFlags) override
virtual HRESULT AddLight (float x, float y, float z, unsigned int lightType, unsigned int color, float
size, float range, bool bAttenuateByAmbient) override
virtual HRESULT BeginLightGroup (ObjectWorldTransform &groupOrigin) override
virtual HRESULT EndLightGroup (bool sortGroup) override
    virtual void ApplyBodyRelativeOffset (const ObjectWorldTransform &llapbhAtOrigin, const
ObjectLocalTransform &offsetXyzPbh, ObjectWorldTransform &llapbhAtOffset)
override
    virtual void CalculateBodyRelativeOffset (const ObjectWorldTransform &llapbhAtOrigin,
const ObjectWorldTransform &llapbhAtOffset, ObjectLocalTransform
&offsetXyzPbh) override
virtual HRESULT CreateDynamicLightData (REFIID riid, void **ppLightData) override
virtual HRESULT AddDynamicLight (IDynamicLightDataV500 *pLightData) override

```

Member Function Documentation

◆ AddDynamicLight()

```
virtual HRESULT AddDynamicLight ( IDynamicLightDataV500 * pLightData ) [private] [pure virtual]
```

Adds a dynamic light to the current view

Parameters

pLightData Light data instance created by CreateDynamicLightData

Returns

Returns S_OK if the light was successfully added to the current view, E_FAIL otherwise

◆ AddLight()

```
virtual HRESULT AddLight ( float           x,
                           float           y,
                           float           z,
                           unsigned int   lightType,
                           unsigned int   color,
                           float           size,
                           float           range,
                           bool            bAttenuateByAmbient
                           )
```

```
[private] [pure virtual]
```

Add a light to the group

Parameters

x	x offset in meters from light group origin
y	y offset in meters from light group origin
z	z offset in meters from light group origin
lightType	Light type
color	Light color
size	Size of light
range	Distance at which light should be visible
bAttenuateByAmbient	Attenuate light based on ambient light in the scene

◆ ApplyBodyRelativeOffset()

```
virtual void  
ApplyBodyRelativeOffset ( const ObjectWorldTransform & llapbhAtOrigin,  
const ObjectLocalTransform & offsetXyzPbh,  
ObjectWorldTransform & llapbhAtOffset  
)
```

private pure virtual

Apply body relative local transformation to a world transform

Parameters

llapbhAtOrigin World transformation of origin or base object
offsetXyzPbh Local body relative transformation to apply as an offset
llapbhAtOffset World transformation resulting from applying the body relative offset

◆ BeginLightGroup()

```
virtual HRESULT BeginLightGroup ( ObjectWorldTransform & groupOrigin ) private pure virtual
```

Begin a light group with a world transform as its origin

Parameters

groupOrigin All lights in the group will be offset relative to this coordinate transformation

◆ CalculateBodyRelativeOffset()

```
virtual void  
CalculateBodyRelativeOffset ( const ObjectWorldTransform & llapbhAtOrigin,  
const ObjectWorldTransform & llapbhAtOffset,  
ObjectLocalTransform & offsetXyzPbh  
)
```

private pure virtual

Calculate body relative offset between two world transforms

Parameters

llapbhAtOrigin World transformation of the origin or base object
llapbhAtOffset World transformation to use as a reference for calculating the offset
offsetXyzPbh Body-relative offset needed to base from the base transform to the reference transform

◆ CreateDynamicLightData()

```
virtual HRESULT CreateDynamicLightData ( REFIID riid,  
void ** ppLightData  
)
```

private pure virtual

Creates an instance of dynamic light data used to add a dynamic light to a view

Parameters

riid The IID of the interface instance to be created (i.e. IID_IDynamicLightDataV500)
ppLightData Pointer to store the newly created instance

Returns

Returns S_OK if the requested interface was successfully created, E_FAIL otherwise

Remarks

The instance is returned with a reference count of 1 and must be released when no longer needed

◆ DrawCylinder()

```
virtual HRESULT  
DrawCylinder ( const ObjectWorldTransform & location,  
float radius,  
float height,
```

```
        ARGBColor  
        RenderFlags  
    )  
    color,  
    renderFlags = 0  
private pure virtual
```

Draw a cylinder

Parameters

location	Location the object will be drawn
radius	Radius of the object in meters
height	Height of the object in meters
color	Color of the object
RenderFlags	Render flags to control drawing

◆ **DrawLine()**

```
virtual HRESULT DrawLine ( const LLADegreesMeters & startLocation,  
                           const LLADegreesMeters & endLocation,  
                           float width,  
                           float height,  
                           ARGBColor color,  
                           RenderFlags renderFlags = 0  
    )  
private pure virtual
```

Draw a line represented by a rectangular prism

Parameters

startLocation	Start location the object will be drawn
endLocation	End location the object will be drawn
width	Width of the object in meters
height	Height of the object in meters
color	Color of the object
RenderFlags	Render flags to control drawing

◆ **DrawRectangle()**

```
virtual HRESULT  
DrawRectangle ( const ObjectWorldTransform & location,  
                float width,  
                float height,  
                float depth,  
                ARGBColor color,  
                RenderFlags renderFlags = 0  
    )  
private pure virtual
```

Draw a rectangular prism

Parameters

location	Location the object will be drawn
width	Width of the object in meters
height	Height of the object in meters
depth	Depth of the object in meters
color	Color of the object
RenderFlags	Render flags to control drawing

◆ **DrawSphere()**

```
virtual HRESULT  
DrawSphere ( const ObjectWorldTransform & location,  
              float radius,  
              ARGBColor color,  
              RenderFlags renderFlags = 0  
    )  
private pure virtual
```

Draw a sphere

Parameters

location Location the object will be drawn
radius Radius of the object in meters
color Color of the object
RenderFlags Render flags to control drawing

◆ DrawText2D()

```
virtual HRESULT DrawText2D ( int x,  
                           int y,  
                           LPCWSTR szText,  
                           ARGBColor textColor,  
                           TextDescription & textDescription,  
                           RenderFlags renderFlags  
 )
```

private pure virtual

Draws text in screen space starting from the top-left with positive directions right and down.

Parameters

x Text location in pixels in the x direction
y Text location in pixels in the y direction
szText Unicode text string to display
textColor Text color
textDescription Text description (i.e. font, alignment, additional flags)
renderFlags Render flags to control drawing

◆ DrawText3D()

```
virtual HRESULT  
DrawText3D ( const ObjectWorldTransform & location,  
             LPCWSTR szText,  
             ARGBColor textColor,  
             TextDescription & textDescription,  
             RenderFlags renderFlags  
 )
```

private pure virtual

Draws text in 3D world space or screen space

Parameters

location Text location (lla and pbh). The text bounding box will be placed at this point.
szText Unicode text string to display
textColor Text color
textDescription Text description (i.e. font, alignment, additional flags)
renderFlags Render flags to control drawing

◆ DrawTriangle()

```
virtual HRESULT  
DrawTriangle ( const ObjectWorldTransform & location,  
               float width,  
               float height,  
               float depth,  
               ARGBColor color,  
               RenderFlags renderFlags = 0  
 )
```

private pure virtual

Draw a triangular prism

Parameters

location Location the object will be drawn
width Width of the object in meters
height Height of the object in meters
depth Depth of the object in meters
color Color of the object

RenderFlags Render flags to control drawing

◆ **EndLightGroup()**

virtual HRESULT EndLightGroup (bool **sortGroup**) [private] [pure virtual]

End a light group.

Parameters

sortGroup If true, this group of lights will be sorted with other transparent objects in the scene. Lights placed up on poles or attached to aircraft should be sorted. Lights placed on the ground generally do not need to be sorted. Unsorted groups are combined into a single draw call for better rendering performance.

◆ **IDynamicLightDataV500**

class **IDynamicLightDataV500**

Interface used to store data needed to create a dynamic light

Inherits **IUnknown**.

Private Member Functions

```
virtual void SetType (DYNAMIC_LIGHT type) override
virtual DYNAMIC_LIGHT GetType () const override
virtual void SetPosition (const ObjectWorldTransform &position)
override
virtual const ObjectWorldTransform & GetPosition () const override
virtual void SetColor (ARGBColor color) override
virtual ARGBColor GetColor () const override
virtual void SetRange (float range) override
virtual float GetRange () const override
virtual void SetOuterAngle (float outerAngle) override
virtual float GetOuterAngle () const override
virtual void SetInnerAngle (float innerAngle) override
virtual float GetInnerAngle () const override
virtual void SetFalloffExponent (float falloffExponent) override
virtual float GetFalloffExponent () const override
virtual void SetIntensity (float intensity) override
virtual float GetIntensity () const override
```

Member Function Documentation

◆ **GetColor()**

virtual **ARGBColor** **GetColor** () const [private] [pure virtual]

Gets the color of the dynamic light

Returns

The diffuse color

Remarks

The alpha value here applies to only non-PBR surfaces, a value of 0 will disable light. 128 is equal to an intensity of 1. Lower than 128 is scaled to an intensity of 0. 255 is equal to an intensity of 50.

◆ **GetFalloffExponent()**

virtual float **GetFalloffExponent** () const [private] [pure virtual]

Gets the falloff exponent of the dynamic light

Returns

The falloff exponent

◆ **GetInnerAngle()**

virtual float GetInnerAngle () const **private** **pure virtual**

Gets the inner angle of the dynamic light

Returns

The inner angle in degrees

◆ **GetIntensity()**

virtual float GetIntensity () const **private** **pure virtual**

Gets the intensity of the dynamic light

Returns

The intensity in lumens

◆ **GetOuterAngle()**

virtual float GetOuterAngle () const **private** **pure virtual**

Gets the outer angle of the dynamic light

Returns

The outer angle in degrees

◆ **GetPosition()**

virtual const **ObjectWorldTransform&** GetPosition () const **private** **pure virtual**

Gets the world position and orientation of the dynamic light

Returns

The world position and orientation

◆ **GetRange()**

virtual float GetRange () const **private** **pure virtual**

Gets the effective range of the dynamic light

Returns

The effective range in meters

◆ **GetType()**

virtual **DYNAMIC_LIGHT** GetType () const **private** **pure virtual**

Gets the type of dynamic light

Returns

The type of dynamic light (i.e. point or spot)

◆ **SetColor()**

virtual void SetColor (**ARGBColor** color) **private** **pure virtual**

Sets the color of the dynamic light

Parameters

color The diffuse color

Remarks

The alpha value here applies to only non-PBR surfaces, a value of 0 will disable light. 128 is equal to an intensity of 1. Lower than 128 is scaled to an intensity of 0. 255 is equal to an intensity of 50.

◆ SetFalloffExponent()

virtual void SetFalloffExponent (float **falloffExponent**) private pure virtual

Sets the falloff exponent of the dynamic light

Parameters

falloffExponent The falloff exponent

◆ SetInnerAngle()

virtual void SetInnerAngle (float **innerAngle**) private pure virtual

Sets the inner angle of the dynamic light

Parameters

innerAngle The inner angle in degrees

◆ SetIntensity()

virtual void SetIntensity (float **intensity**) private pure virtual

Sets the intensity of the dynamic light

Parameters

intensity The intensity in lumens

◆ SetOuterAngle()

virtual void SetOuterAngle (float **outerAngle**) private pure virtual

Sets the outer angle of the dynamic light

Parameters

outerAngle The outer angle in degrees

◆ SetPosition()

virtual void SetPosition (const **ObjectWorldTransform** & **position**) private pure virtual

Sets the world position and orientation of the dynamic light

Parameters

position The world position and orientation

◆ SetRange()

virtual void SetRange (float **range**) private pure virtual

Sets the effective range of the dynamic light

Parameters

range The effective range in meters

◆ SetType()

virtual void SetType (**DYNAMIC_LIGHT** type) [private] [pure virtual]

Sets the type of dynamic light

Parameters

type The type of dynamic light (i.e. point or spot)

◆ P3D::RenderingPlugin

class P3D::RenderingPlugin

Base implementation of the **IRenderingPluginV500** which defines default implementations of all **IRenderingPluginV500** functions and a default IUnknown implementation.

Inherits **IRenderingPluginV500**.

Public Member Functions

RenderingPlugin () noexcept

 virtual ~**RenderingPlugin** ()

 virtual void **PreRender** (**IRenderDataV500** *pRenderData)

 virtual void **Render** (**IRenderDataV500** *pRenderData) override

DEFAULT_REFCOUNT_INLINE_IMPL ()

 STDMETHODIMP **QueryInterface** (REFIID riid, **PVOID** *ppv)

Protected Attributes

PdkRenderFlags mRenderFlags

Constructor & Destructor Documentation

◆ RenderingPlugin()

RenderingPlugin () [inline] [noexcept]

◆ ~RenderingPlugin()

virtual ~**RenderingPlugin** () [inline] [virtual]

Member Function Documentation

◆ DEFAULT_REFCOUNT_INLINE_IMPL()

DEFAULT_REFCOUNT_INLINE_IMPL ()

◆ PreRender()

virtual void **PreRender** (**IRenderDataV500** * pRenderData) [inline] [virtual]

Enables rendering if the current render pass is RenderPassDefault

Implements **IRenderingPluginV500**.

◆ QueryInterface()

STDMETHODIMP **QueryInterface** (REFIID riid,
 PVOID * ppv

)

inline

◆ **Render()**

virtual void Render (**IRenderDataV500** * pRenderData) **pure virtual**

A callback to render content

Parameters

pRenderData Interface to rendering device and resources used for rendering.

Remarks

Render will be called during each rendering pass unless the RenderingIsEnabled flag is set to false in PreRender.

Implements **IRenderingPluginV500**.

Member Data Documentation

◆ **mRenderFlags**

PdkRenderFlags mRenderFlags **protected**

Macros

```
#define RS_IPLUGIN_SYSTEM_LEGACY_H
```

TypeDefs

```
typedef IRenderDataV500 IRenderData
typedef IRenderDataSourceV500 IRenderDataSource
typedef IRenderingPluginV500 IRenderingPlugin
typedef IRenderingPluginSystemV500 IRenderingPluginSystem
```

Enumerations

```
enum PdkRenderPass {
    RenderPassPreVc = 0, RenderPassPreVcPanels, RenderPassDefault,
    RenderPassPreScene,
    RenderPassPrimaryThreadUpdate
}
```

Variables

```
REFIID IID_IRenderingPluginSystemV400 = __uuidof(IRenderingPluginSystemV400)
REFIID IID_IRenderingPluginSystemV430 = __uuidof(IRenderingPluginSystemV430)
REFIID IID_IRenderingPluginSystemV440 = __uuidof(IRenderingPluginSystemV440)
REFIID IID_IRenderDataV400 = __uuidof(IRenderDataV400)
REFIID IID_IRenderDataV430 = __uuidof(IRenderDataV430)
REFIID IID_IRenderingPluginV400 = __uuidof(IRenderingPluginV400)
REFIID IID_IRenderingPluginSystemV500 = __uuidof(IRenderingPluginSystemV500)
REFGUID SID_RenderingPluginSystem = __uuidof(IRenderingPluginSystemV400)
REFIID IID_IRenderingPluginV500 = __uuidof(IRenderingPluginV500)
REFIID IID_IRenderDataV500 = __uuidof(IRenderDataV500)
REFIID IID_IRenderDataResourceV500 = __uuidof(IRenderDataResourceV500)
REFIID IID_IRenderingPlugin = IID_IRenderingPluginV500
REFIID IID_IRenderData = IID_IRenderDataV500
REFIID IID_IRenderDataResource = IID_IRenderDataResourceV500
REFIID IID_IRenderingPluginSystem = IID_IRenderingPluginSystemV500
REFIID IID_IObjectRendererV500 = __uuidof(IObjectRendererV500)
REFIID IID_IDynamicLightDataV500 = __uuidof(IDynamicLightDataV500)
REFIID SID_ObjectRenderer = IID_IObjectRendererV400
```

Macro Definition Documentation

◆ RS_IPLUGIN_SYSTEM_LEGACY_H

```
#define RS_IPLUGIN_SYSTEM_LEGACY_H
```

Typedef Documentation

◆ IRenderData

```
typedef IRenderDataV500 IRenderData
```

◆ IRenderDataResource

```
typedef IRenderDataResourceV500 IRenderDataResource
```

◆ IRenderingPlugin

```
typedef IRenderingPluginV500 IRenderingPlugin
```

◆ IRenderingPluginSystem

```
typedef IRenderingPluginSystemV500 IRenderingPluginSystem
```

Enumeration Type Documentation

◆ PdkRenderPass

```
enum PdkRenderPass
```

Enumerator

RenderPassPreVc

RenderPassPreVcPanels

RenderPassDefault

RenderPassPreScene

RenderPassPrimaryThreadUpdate

Variable Documentation

◆ IID_IDynamicLightDataV500

```
REFIID IID_IDynamicLightDataV500 = __uuidof(IDynamicLightDataV500)
```

◆ IID_IObjectRendererV500

```
REFIID IID_IObjectRendererV500 = __uuidof(IObjectRendererV500)
```

◆ IID_IRenderData

```
REFIID IID_IRenderData = IID_IRenderDataV500
```

◆ IID_IRenderDataSource

```
REFIID IID_IRenderDataSource = IID_IRenderDataSourceV500
```

◆ IID_IRenderDataSourceV500

REFIID IID_IRenderDataSourceV500 = __uuidof(IRenderDataSourceV500)

◆ IID_IRenderDataV400

REFIID IID_IRenderDataV400 = __uuidof(IRenderDataV400)

◆ IID_IRenderDataV430

REFIID IID_IRenderDataV430 = __uuidof(IRenderDataV430)

◆ IID_IRenderDataV500

REFIID IID_IRenderDataV500 = __uuidof(IRenderDataV500)

◆ IID_IRenderingPlugin

REFIID IID_IRenderingPlugin = IID_IRenderingPluginV500

◆ IID_IRenderingPluginSystem

REFIID IID_IRenderingPluginSystem = IID_IRenderingPluginSystemV500

◆ IID_IRenderingPluginSystemV400

REFIID IID_IRenderingPluginSystemV400 = __uuidof(IRenderingPluginSystemV400)

◆ IID_IRenderingPluginSystemV430

REFIID IID_IRenderingPluginSystemV430 = __uuidof(IRenderingPluginSystemV430)

◆ IID_IRenderingPluginSystemV440

REFIID IID_IRenderingPluginSystemV440 = __uuidof(IRenderingPluginSystemV440)

◆ IID_IRenderingPluginSystemV500

REFIID IID_IRenderingPluginSystemV500 = __uuidof(IRenderingPluginSystemV500)

◆ IID_IRenderingPluginV400

REFIID IID_IRenderingPluginV400 = __uuidof(IRenderingPluginV400)

◆ IID_IRenderingPluginV500

REFIID IID_IRenderingPluginV500 = __uuidof(IRenderingPluginV500)

◆ SID_ObjectRenderer

REFIID SID_ObjectRenderer = IID_IObjectRendererV400

◆ SID_RenderingPluginSystem

REFGUID SID_RenderingPluginSystem = __uuidof(IRenderingPluginSystemV400)

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Weather System Service

Overview

This PDK service allows callers to manipulate the weather system during runtime. The interface gives the developer control over current weather settings as well as current weather station data.

Classes

class [IWeatherStationV430](#)

class [IWeatherSystemV500](#)

Class Documentation

◆ [P3D::IWeatherStationV430](#)

class [P3D::IWeatherStationV430](#)

Inherits [IUnknown](#).

Public Member Functions

virtual bool [IsValid \(\) const override](#)

virtual float [GetSurfaceWind \(\) const override](#)

virtual HRESULT [SetSurfaceWind \(float aVal\) const override](#)

virtual float [GetBaroPressure \(\) const override](#)

virtual HRESULT [SetBaroPressure \(float aVal\) const override](#)

virtual int [GetCloudLayerCount \(\) const override](#)

virtual HRESULT [GetCloudAtIndex \(_in UINT uIndex, P3D::CloudLayer &aLayer\) const override](#)

virtual HRESULT [AddNewCloudToTower \(_in CloudLayer &aLayer\) const override](#)

```
virtual HRESULT RemoveCloudLayerFromTower (int ulIndex) const override
virtual HRESULT ClearAllCloudsAtTower () const override
    virtual UINT GetTempLayerCount () const override
virtual HRESULT GetTempLayerAtIndex (__in UINT ulIndex, __out P3D::TempLayer &aLayer) const override
virtual HRESULT AddNewTempToTower (__in TempLayer &aLayer) const override
virtual HRESULT RemoveTempLayerFromTower (int ulIndex) const override
virtual HRESULT ClearAllTempLayersFromTower () const override
    virtual UINT GetAloftLayerCount () const override
virtual HRESULT GetAloftLayerAtIndex (__in UINT ulIndex, __out P3D::WindAloftLayer &aLayer) const override
virtual HRESULT AddNewAloftToTower (__in WindAloftLayer &aLayer) const override
virtual HRESULT RemoveAloftLayerFromTower (int ulIndex) const override
virtual HRESULT ClearAllWindLayersFromTower () const override
    virtual UINT GetVisibilityLayerCount () const override
virtual HRESULT GetVisibilityLayerAtIndex (__in UINT ulIndex, __out P3D::VisibilityLayer &aLayer) const override
virtual HRESULT AddNewVisToTower (__in VisibilityLayer &aLayer) const override
virtual HRESULT RemoveVisLayerFromTower (int ulIndex) const override
virtual HRESULT ClearAllVisLayersFromTower () const override
```

Member Function Documentation

◆ AddNewAloftToTower()

```
virtual HRESULT AddNewAloftToTower ( __in WindAloftLayer & aLayer ) const [pure virtual]
```

◆ AddNewCloudToTower()

```
virtual HRESULT AddNewCloudToTower ( __in CloudLayer & aLayer ) const [pure virtual]
```

◆ AddNewTempToTower()

```
virtual HRESULT AddNewTempToTower ( __in TempLayer & aLayer ) const [pure virtual]
```

◆ AddNewVisToTower()

```
virtual HRESULT AddNewVisToTower ( __in VisibilityLayer & aLayer ) const [pure virtual]
```

◆ ClearAllCloudsAtTower()

virtual HRESULT ClearAllCloudsAtTower() const pure virtual

◆ **ClearAllTempLayersFromTower()**

virtual HRESULT ClearAllTempLayersFromTower() const pure virtual

◆ **ClearAllVisLayersFromTower()**

virtual HRESULT ClearAllVisLayersFromTower() const pure virtual

◆ **ClearAllWindLayersFromTower()**

virtual HRESULT ClearAllWindLayersFromTower() const pure virtual

◆ **GetAloftLayerAtIndex()**

virtual HRESULT GetAloftLayerAtIndex(__in UINT ulIndex,
 __out P3D::WindAloftLayer & aLayer
)
 const pure virtual

◆ **GetAloftLayerCount()**

virtual UINT GetAloftLayerCount() const pure virtual

◆ **GetBaroPressure()**

virtual float GetBaroPressure() const pure virtual

◆ **GetCloudAtIndex()**

virtual HRESULT GetCloudAtIndex(__in UINT ulIndex,
 P3D::CloudLayer & aLayer
)
 const pure virtual

◆ **GetCloudLayerCount()**

virtual int GetCloudLayerCount() const pure virtual

◆ **GetSurfaceWind()**

virtual float GetSurfaceWind() const pure virtual

◆ GetTempLayerAtIndex()

```
virtual HRESULT GetTempLayerAtIndex ( __in UINT ulIndex,
                                     __out P3D::TempLayer & aLayer
                                     )
                                     const pure virtual
```

◆ GetTempLayerCount()

```
virtual UINT GetTempLayerCount ( ) const pure virtual
```

◆ GetVisibilityLayerAtIndex()

```
virtual HRESULT
GetVisibilityLayerAtIndex ( __in UINT ulIndex,
                           __out P3D::VisibilityLayer & aLayer
                           )
                           const pure virtual
```

◆ GetVisibilityLayerCount()

```
virtual UINT GetVisibilityLayerCount ( ) const pure virtual
```

◆ IsValid()

```
virtual bool IsValid ( ) const pure virtual
```

◆ RemoveAloftLayerFromTower()

```
virtual HRESULT RemoveAloftLayerFromTower ( int ulIndex ) const pure virtual
```

◆ RemoveCloudLayerFromTower()

```
virtual HRESULT RemoveCloudLayerFromTower ( int ulIndex ) const pure virtual
```

◆ RemoveTempLayerFromTower()

```
virtual HRESULT RemoveTempLayerFromTower ( int ulIndex ) const pure virtual
```

◆ RemoveVisLayerFromTower()

```
virtual HRESULT RemoveVisLayerFromTower ( int ulIndex ) const pure virtual
```

◆ **SetBaroPressure()**

```
virtual HRESULT SetBaroPressure ( float aVal ) const [pure virtual]
```

◆ **SetSurfaceWind()**

```
virtual HRESULT SetSurfaceWind ( float aVal ) const [pure virtual]
```

◆ **P3D::IWeatherSystemV500**

class P3D::IWeatherSystemV500

This is the interface to the core Prepar3D weather system. An example on how to query for the service can be found in the DLLStart() function of the Camera PDK Sample.

Remarks

This interface uses several enumerated weather types which are defined in **WeatherSystemTypes.h**. Several functions will also utilize the METAR Data Format.

Inherits IWeatherSystemV430.

Public Member Functions

virtual bool **GetEnhancedAtmospherics () const override**
Get the current enhanced atmospherics setting. [More...](#)

virtual HRESULT **SetEnhancedAtmospherics (_in bool bEnable)**
override
Set the current enhanced atmospherics setting. [More...](#)

virtual **CLOUD_COVERAGE_DENSITY** **GetCloudCoverageDensity () const override**
Get the current cloud coverage density. [More...](#)

virtual HRESULT **SetCloudCoverageDensity (_in CLOUD_COVERAGE_DENSITY eDensity)**
override

virtual **CLOUD_DRAW_DISTANCE** **GetCloudDrawDistance () const override**
Get the current cloud draw distance. [More...](#)

virtual HRESULT **SetCloudDrawDistance (_in CLOUD_DRAW_DISTANCE eDistance) override**
Set the current cloud draw distance. [More...](#)

virtual bool **GetDetailedClouds () const override**
Get the detailed cloud setting. [More...](#)

virtual HRESULT **SetDetailedClouds (_in bool bIsDetailed) override**
Set the detailed cloud setting. [More...](#)

virtual **THERMAL_VISUAL_TYPE** **GetThermalVisualType () const override**
Get the current thermal visual type. [More...](#)

virtual HRESULT **SetThermalVisualType (_in**

	THERMAL_VISUAL_TYPE eThermalType)
	override
	Set the current thermal visual type. More...
virtual int	GetNumberOfPressurePoints () const override
	Get the current number of pressure points. More...
virtual HRESULT	GetStationMetarData (<u>in</u> __notnull LPCSTR pszIcao, <u>out</u> __notnull LPWSTR pszMetar, <u>in</u> size_t cchMetar) const override
virtual HRESULT	GetNearestStationMetarData (<u>in</u> double dLatRadians, <u>in</u> double dLonRadians, <u>in</u> double dAltFeet, <u>out</u> __notnull LPWSTR pszMetar, <u>in</u> size_t cchMetar) const override
virtual HRESULT	GetInterpolatedMetarData (<u>in</u> double dLatRadians, <u>in</u> double dLonRadians, <u>in</u> double dAltFeet, <u>out</u> __notnull LPWSTR pszMetar, <u>in</u> size_t cchMetar) const override
virtual HRESULT	SetMetarData (<u>in</u> __notnull LPCWSTR pszMetar, <u>in</u> int nSeconds) override
virtual HRESULT	SetWeatherMode (<u>in</u> WEATHER_MODE eWeatherMode, <u>in_opt</u> LPCWSTR pszThemePath=nullptr) override
virtual WEATHER_MODE	GetWeatherMode () const override
	Gets the current weather mode. More...
virtual HRESULT	GetThemePath (<u>out</u> __notnull LPWSTR pszThemePath, <u>in</u> size_t cchThemePath) const override
virtual HRESULT	CreateStation (<u>in</u> __notnull LPCWSTR pszICAO, <u>in</u> __notnull LPCWSTR pszName, <u>in</u> double dLatRadians, <u>in</u> double dLonRadians, <u>in</u> double dAltFeet, <u>out</u> __notnull void **ppStation) override
virtual HRESULT	RemoveStation (<u>in</u> __notnull void *pStation) override
virtual HRESULT	CreateThermal (<u>in</u> double dLatRadians, <u>in</u> double dLonRadians, <u>in</u> double dAltFeet, <u>in</u> float fRadiusMeters, <u>in</u> float fHeightMeters, <u>in</u> float fCoreRateMps, <u>in</u> float fCoreTurbulenceMps, <u>in</u> float fSinkRateMps, <u>in</u> float fSinkTurbulenceMps, <u>in</u> float fCoreSizeMeters, <u>in</u> float fCoreTransitionSizeMeters, <u>in</u> float fSinkLayerSizeMeters, <u>in</u> float fSinkTransitionSizeMeters, <u>out</u> __notnull void **ppThermal) override
virtual HRESULT	RemoveThermal (<u>in</u> __notnull void *pThermal) override
virtual HRESULT	RequestCloudState (<u>in</u> double dLatRadiansMin, <u>in</u> double dLonRadiansMin, <u>in</u> double dAltFeetMin, <u>in</u> double dLatRadiansMax, <u>in</u> double dLonRadiansMax, <u>in</u> double dAltFeetMax, <u>in</u> uint cbCloudState, <u>out</u> __notnull BYTE *pCloudState) const override

	virtual HRESULT	SetDynamicUpdateRate (<u>in</u> DWORD dwRate) override
	virtual float	GetGlobalTemp () const override Get the global temp in degrees C. More...
	virtual HRESULT	SetGlobalTemp (<u>in</u> float aVal) override Set the global temp using degrees C. More...
	virtual float	GetGlobalVisRange () const override Get the global visibility range in Meters. More...
	virtual HRESULT	SetGlobalVisRange (<u>in</u> float aVal) override Set the global visibility range in Meters. More...
	virtual float	GetGlobalHorizWindSpeed () const override Get the global horizontal wind speed in m/s. More...
	virtual HRESULT	SetGlobalHorizWindSpeed (<u>in</u> float aVal) override Set the global horizontal wind speed in knots. More...
	virtual float	GetGlobalWindDirection () const override Get the global wind direction in degrees. More...
	virtual HRESULT	SetGlobalWindDirection (<u>in</u> float aVal) override Set the global wind direction in degrees. More...
	virtual float	GetGlobalBaroPressure () const override Get the global barometer pressure in millibars. More...
	virtual HRESULT	SetGlobalBaroPressure (<u>in</u> float aVal) override Set the global barometer pressure in millibars. More...
	virtual float	GetGlobalDewPoint () const override Get the global dew point. More...
	virtual HRESULT	SetGlobalDewPoint (<u>in</u> float aVal) override Set the global dew point. More...
	virtual HRESULT	GetWeatherStation (<u>in</u> LPCSTR pszICAO, <u>out</u> P3D::IWeatherStationV430 **ppStation) override
	virtual HRESULT	ReloadWeather () override

Member Function Documentation

◆ CreateStation()

```
virtual HRESULT CreateStation ( in __notnull LPCWSTR pszICAO,
                               in __notnull LPCWSTR pszName,
                               in double dLatRadians,
                               in double dLonRadians,
                               in double dAltFeet,
                               out __notnull void ** ppStation
                           )
```

pure virtual

Creates a weather station at the given ICAO location

Parameters

- IN** pszICAO The ICAO of the station to be created.
- IN** pszName A descriptive name for the station. This name will appear on weather maps and within dialogs.
- IN** dLatRadians The latitude of the station in radians.
- IN** dLonRadians The longitude of the station in radians.
- IN** dAltFeet The altitude of the station in feet.
- OUT** ppStation The reference pointer to be used by [RemoveStation\(\)](#).

◆ CreateThermal()

```
virtual HRESULT  
CreateThermal( _in double dLatRadians,  
               _in double dLonRadians,  
               _in double dAltFeet,  
               _in float fRadiusMeters,  
               _in float fHeightMeters,  
               _in float fCoreRateMps,  
               _in float fCoreTurbulenceMps,  
               _in float fSinkRateMps,  
               _in float fSinkTurbulenceMps,  
               _in float fCoreSizeMeters,  
               _in float fCoreTransitionSizeMeters,  
               _in float fSinkLayerSizeMeters,  
               _in float fSinkTransitionSizeMeters,  
               __out __notnull void ** ppThermal  
) pure virtual
```

Creates a thermal at the given location

Parameters

- IN** dLatRadians The latitude of the thermal in radians.
- IN** dLonRadians The longitude of the thermal in radians.
- IN** dAltFeet The altitude of the thermal in feet.
- IN** fRadiusMeters Specifies the radius of the thermal, in meters. The maximum radius of a thermal is 100 km.
- IN** fHeightMeters Specifies the height of the thermal, in meters.
- IN** fCoreRateMps Specifies the lift value, in meters per second, within the Core layer. A positive value will provide an updraft, a negative value a downdraft. The maximum rate is 1000 meters/second.
- IN** fCoreTurbulenceMps Specifies a variation in meters per second that is applied to the coreRate. For example, if a value of 1.5 is entered, and the core rate is 5 m/s, the actual core rate applied will be randomly varying between 3.5 m/s and 6.5 m/s.
- IN** fSinkRateMps Specifies the lift value, in meters per second, within the Sink layer. A positive value will provide an updraft, a negative value a downdraft. The maximum rate is 1000 meters/second.
- IN** fSinkTurbulenceMps Specifies a variation in meters per second that is applied to the sinkRate. For example, if a value of 1.5 is entered, and the sink rate is 5 m/s, the actual sink rate applied will be randomly varying between 3.5 m/s and 6.5 m/s.

- IN** fCoreSizeMeters Specifies the radius in meters of the Core of the thermal.
- IN** fCoreTransitionSizeMeters Specifies the width in meters of the transition layer between the Core and the Sink of the thermal. Half of the width of this transition will be outside the Core, and half within.
- IN** fSinkLayerSizeMeters Specifies the radius in meters of the Sink of the thermal.
- IN** fSinkTransitionSizeMeters Specifies the width in meters of the transition layer between the Sink and the atmosphere outside of the thermal. Half of the width of this transition will be outside the radius of the Sink layer, and half within.
- OUT** ppThermal The reference pointer to be used by [RemoveThermal\(\)](#).

◆ **GetCloudCoverageDensity()**

virtual **CLOUD_COVERAGE_DENSITY** GetCloudCoverageDensity() const **pure virtual**

Get the current cloud coverage density.

◆ **GetCloudDrawDistance()**

virtual **CLOUD_DRAW_DISTANCE** GetCloudDrawDistance() const **pure virtual**

Get the current cloud draw distance.

◆ **GetDetailedClouds()**

virtual bool GetDetailedClouds() const **pure virtual**

Get the detailed cloud setting.

◆ **GetEnhancedAtmospherics()**

virtual bool GetEnhancedAtmospherics() const **pure virtual**

Get the current enhanced atmospherics setting.

◆ **GetGlobalBaroPressure()**

virtual float GetGlobalBaroPressure() const **pure virtual**

Get the global barometer pressure in millibars.

◆ **GetGlobalDewPoint()**

virtual float GetGlobalDewPoint() const **pure virtual**

Get the global dew point.

◆ **GetGlobalHorizWindSpeed()**

virtual float GetGlobalHorizWindSpeed() const **pure virtual**

Get the global horizontal wind speed in m/s.

◆ **GetGlobalTemp()**

virtual float GetGlobalTemp() const **pure virtual**

Get the global temp in degrees C.

◆ **GetGlobalVisRange()**

virtual float GetGlobalVisRange() const **pure virtual**

Get the global visibility range in Meters.

◆ **GetGlobalWindDirection()**

virtual float GetGlobalWindDirection() const **pure virtual**

Get the global wind direction in degrees.

◆ **GetInterpolatedMetarData()**

```
virtual HRESULT  
GetInterpolatedMetarData( __in double dLatRadians,  
                           __in double dLonRadians,  
                           __in double dAltFeet,  
                           __out __notnull LPWSTR pszMetar,  
                           __in size_t cchMetar  
                           ) const pure virtual
```

Get the weather station interpolated METAR string at the given latitude, longitude, and altitude. The maximum supported length for a METAR string is 2000 characters. See also the METAR Data Format.

◆ **GetNearestStationMetarData()**

virtual HRESULT

```
GetNearestStationMetarData ( __in double dLatRadians,  
                           __in double dLonRadians,  
                           __in double dAltFeet,  
                           __out __notnull LPWSTR pszMetar,  
                           __in size_t cchMetar  
                           ) const pure virtual
```

Get the METAR string from the nearest weather station at the given latitude, longitude, and altitude. The maximum supported length for a METAR string is 2000 characters. See also the METAR Data Format.

◆ GetNumberOfPressurePoints()

```
virtual int GetNumberOfPressurePoints ( ) const pure virtual
```

Get the current number of pressure points.

◆ GetStationMetarData()

```
virtual HRESULT GetStationMetarData ( __in __notnull LPCSTR pszIcao,  
                                     __out __notnull LPWSTR pszMetar,  
                                     __in size_t cchMetar  
                                     ) const pure virtual
```

Get the METAR string from the weather station with the given ICAO. The maximum supported length for a METAR string is 2000 characters. See also the METAR Data Format.

◆ GetThemePath()

```
virtual HRESULT GetThemePath ( __out __notnull LPWSTR pszThemePath,  
                             __in size_t cchThemePath  
                             ) const pure virtual
```

Gets the current weather theme file path.

Parameters

pszThemePath OUT: The output string to store the theme file path.
cchThemePath IN: The length of the pszThemePath parameter in characters.

Remarks

Succeeding and returning an empty string represents clear weather.

◆ GetThermalVisualType()

```
virtual THERMAL_VISUAL_TYPE GetThermalVisualType ( ) const pure virtual
```

Get the current thermal visual type.

◆ GetWeatherMode()

virtual WEATHER_MODE GetWeatherMode() const pure virtual

Gets the current weather mode.

◆ GetWeatherStation()

virtual HRESULT GetWeatherStation

```
( __in LPCSTR pszIcao,
  __out P3D::IWeatherStationV430 ** ppStation
)
```

pure virtual

◆ ReloadWeather()

virtual HRESULT ReloadWeather() pure virtual

◆ RemoveStation()

virtual HRESULT RemoveStation (__in __notnull void * pStation) pure virtual

Removes the weather station with the given reference pointer.

Parameters

IN pStation The reference pointer returned by [CreateStation\(\)](#).

◆ RemoveThermal()

virtual HRESULT RemoveThermal (__in __notnull void * pThermal) pure virtual

Removes a thermal with the given reference pointer.

Parameters

pThermal The reference pointer returned by [CreateThermal\(\)](#).

◆ RequestCloudState()

```
    __in double dAltFeetMax,  
    __in UINT cbCloudState,  
    __out __notnull BYTE * pCloudState  
) const pure virtual
```

Returns the cloud state for the given area.

Parameters

- IN** dLatRadiansMin Specifies the minimum latitude of the required area. This should simply be the lower of the two latitude numbers.
- IN** dLonRadiansMin Specifies the minimum longitude of the required area. This should simply be the lower of the two longitude numbers.
- IN** dAltFeetMin Specifies the minimum altitude of the required area, in feet.
- IN** dLatRadiansMax Specifies the maximum latitude of the required area.
- IN** dLonRadiansMax Specifies the maximum longitude of the required area.
- IN** dAltFeetMax Specifies the maximum altitude of the required area, in feet.
- IN** cbCloudState The size of the cloud state byte array. This value must be 64 * 64 (4096).
- OUT** pCloudState The byte array to receive the cloud state data. This buffer must be 64 * 64 (4096) bytes. Cell data within this array is structured as a 64 x 64 grid. A cell value of zero would mean no cloud layers, to a maximum of 255 layers.

Remarks

Note that the entire world's weather is not simulated all the time. Only a region around the user aircraft with a radius of approximately 128 kilometers is modeled at any one time. A request for cloud data outside this region will simply return zeros. The defined area can cross the Equator or the Greenwich Meridian, but it cannot cross the Poles or the International Date Line.

◆ SetCloudCoverageDensity()

```
virtual HRESULT SetCloudCoverageDensity( __in CLOUD_COVERAGE_DENSITY eDensity ) pure virtual
```

Set the current cloud coverage density. The effects of setting this value will not be immediately seen.

◆ SetCloudDrawDistance()

```
virtual HRESULT SetCloudDrawDistance( __in CLOUD_DRAW_DISTANCE eDistance ) pure virtual
```

Set the current cloud draw distance.

◆ SetDetailedClouds()

```
virtual HRESULT SetDetailedClouds( __in bool bIsDetailed ) pure virtual
```

Set the detailed cloud setting.

◆ **SetDynamicUpdateRate()**

virtual HRESULT SetDynamicUpdateRate (__in DWORD dwRate) **pure virtual**

Sets the rate at which cloud formations change.

Parameters

Double word containing the rate. A value of zero indicates that cloud formations do not change at all. Values between 1 and 5 indicate that cloud formations should change from 1 (the slowest) to 5 (the fastest).

◆ **SetEnhancedAtmospherics()**

virtual HRESULT SetEnhancedAtmospherics (__in bool bEnable) **pure virtual**

Set the current enhanced atmospherics setting.

◆ **SetGlobalBaroPressure()**

virtual HRESULT SetGlobalBaroPressure (__in float aVal) **pure virtual**

Set the global barometer pressure in millibars.

◆ **SetGlobalDewPoint()**

virtual HRESULT SetGlobalDewPoint (__in float aVal) **pure virtual**

Set the global dew point.

◆ **SetGlobalHorizWindSpeed()**

virtual HRESULT SetGlobalHorizWindSpeed (__in float aVal) **pure virtual**

Set the global horizontal wind speed in knots.

◆ **SetGlobalTemp()**

virtual HRESULT SetGlobalTemp (__in float aVal) **pure virtual**

Set the global temp using degrees C.

◆ SetGlobalVisRange()

```
virtual HRESULT SetGlobalVisRange ( __in float aVal ) pure virtual
```

Set the global visibility range in Meters.

◆ SetGlobalWindDirection()

```
virtual HRESULT SetGlobalWindDirection ( __in float aVal ) pure virtual
```

Set the global wind direction in degrees.

◆ SetMetarData()

```
virtual HRESULT SetMetarData ( __in __notnull LPCWSTR pszMetar,
                               __in int nSeconds
                           ) pure virtual
```

Set the weather system METAR string. The maximum supported length for a METAR string is 2000 characters. See also the METAR Data Format.

◆ SetThermalVisualType()

```
virtual HRESULT SetThermalVisualType ( __in THERMAL_VISUAL_TYPE eThermalType ) pure virtual
```

Set the current thermal visual type.

◆ SetWeatherMode()

```
virtual HRESULT SetWeatherMode ( __in WEATHER_MODE eWeatherMode,
                                 __in_opt LPCWSTR pszThemePath = nullptr
                           ) pure virtual
```

Sets the current weather mode.

Parameters

eWeatherMode IN: The WEATHER_MODE to be set.

pszThemePath IN: Null-terminated string containing the theme path and filename. The theme file path is only used by the WEATHER_MODE_THEME type. The path can be either:

1. An absolute path (e.g. C:\Program Files\My Company\My Product\myweathertheme)
2. A path relative to the Lockheed Martin\Prepar3D v5\ installation

- folder (e.g. weather\themes\grayrain)
3. A path relative to any of the weather theme folders found in any of the weather.cfg configuration files (e.g. grayrain)
 4. Null or an empty string to set clear weather

Variables

GUID **IID_IWeatherSystemV500**

WeatherSystemPdk interface ID. [More...](#)

GUID **SID_WeatherSystem**

WeatherSystemPdk service ID. [More...](#)

GUID **IID_IWeatherStationV430**

WeatherSystemPdk. [More...](#)

Variable Documentation

◆ **IID_IWeatherStationV430**

GUID **IID_IWeatherStationV430**

WeatherSystemPdk.

◆ **IID_IWeatherSystemV500**

GUID **IID_IWeatherSystemV500**

WeatherSystemPdk interface ID.

◆ **SID_WeatherSystem**

GUID **SID_WeatherSystem**

WeatherSystemPdk service ID.

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Visual Effect Service

Overview

This PDK service allows callers to manipulate visual effects during runtime. Effects spawned using this interface are defined with a world-based position and are not associated with any other object.

Classes

- class **IVisualEffectManagerV430**
class **IVisualEffectCallbackV430**
class **IRopeSimulationV420**
-

Class Documentation

◆ P3D::IVisualEffectManagerV430

class P3D::IVisualEffectManagerV430

This PDK service allows callers to manipulate visual effects during runtime in the following ways:

- Turn effects off and on
- Modify the position and orientation of an effect.
- Modify the duration of an active effect.

Remarks

Effects spawned using this interface are defined with a world-based position and are not associated with any other object.

Inherits IVisualEffectManagerV410.

Private Member Functions

- virtual HRESULT **EffectOn** (_in _notnull LPCWSTR pszEffectName, _in const **P3DDXYZ** &vWorldPosRadiansFeet, _in const **P3DDXYZ** &vOrientationRadians, _in BOOL bOnGround, _out int &iEffectID) **PURE**
virtual HRESULT **EffectOff** (_inout int &iEffectID) **PURE**
virtual HRESULT **SetEffectDuration** (_in int iEffectID, float fDurationSecs) **PURE**
virtual HRESULT **SetEffectWorldPosRadiansFeet** (_in int iEffectID, _in const **P3DDXYZ** &vWorldPosRadiansFeet) **PURE**
virtual HRESULT **SetEffectWorldOrientationRadians** (_in int iEffectID, _in const **P3DDXYZ** &vOrientationRadians) **PURE**
virtual HRESULT **CreateRopeSimulation** (_in const **P3DFXYZ** &vStart, _in const **P3DFXYZ** &vEnd, _in float fRopeMass, _in float fRestingRopeLength, _in float fRelativeGroundPosition, _in REFIID iid, _out void **ppvRopeSim) **PURE**
virtual HRESULT **CreateRopeSimulation** (_in const **P3DFXYZ** &vStart, _in const **P3DFXYZ** &vEnd, _in float fRopeMass, _in float fRestingRopeLength, _in float fRelativeGroundPosition, _out IRopeSimulationV400 **ppvRopeSim) **PURE**
virtual HRESULT **RegisterVisualEffectCallback** (IVisualEffectCallbackV430 *pfCallback)
override
virtual HRESULT **UnRegisterVisualEffectCallback** (IVisualEffectCallbackV430 *pfCallback)
override
virtual HRESULT **GetEffectWorldPosRadiansFeet** (_in HANDLE hEffect, _out **P3DDXYZ** &vWorldPosRadiansFeet) **PURE**
virtual HRESULT **GetEffectWorldOrientationRadians** (_in HANDLE hEffect, _out **P3DDXYZ** &vOrientationRadians) **PURE**

Member Function Documentation

◆ CreateRopeSimulation() [1 / 2]

```
virtual HRESULT  
CreateRopeSimulation( __in const P3DFXYZ & vStart,  
                      __in const P3DFXYZ & vEnd,  
                      __in float          fRopeMass,  
                      __in float          fRestingRopeLength,  
                      __in float          fRelativeGroundPosition,  
                      __in REFIID          riid,  
                      __out void **        ppvIRopeSim  
)
```

private virtual

Creates a dynamic "rope" visual effect. This can be used to attach physics, such as a hoisted load.

Parameters

vStart	Initial offset of start of rope, in feet.
vEnd	Initial offset of end of rope, in feet.
fRopeMass	Mass of rope, in slugs.
fRestingRopeLength	Unstretched length of rope, in feet.
fRelativeGroundPosition	Relative height of the ground, in feet. Typically negative.
riid	Interface ID for the version of the IRopeSimulation being requested.
ppvIRopeSim	Returns reference-counted rope simulation which can be updated in real-time through the IRopeSimulation interface (below).

◆ CreateRopeSimulation() [2 / 2]

```
virtual HRESULT  
CreateRopeSimulation( __in const P3DFXYZ &           vStart,  
                      __in const P3DFXYZ &           vEnd,  
                      __in float              fRopeMass,  
                      __in float              fRestingRopeLength,  
                      __in float              fRelativeGroundPosition,  
                      __out IRopeSimulationV400 ** ppIRopeSim  
)
```

private virtual

Deprecated version of CreateRopeSimulation.

◆ EffectOff()

```
virtual HRESULT EffectOff( __inout int & iEffectID ) private virtual
```

Turn effect off

Parameters

iEffectID Effect id

◆ EffectOn()

```
virtual HRESULT EffectOn( __in __notnull LPCWSTR pszEffectName,  
                          __in const P3DDXYZ &           vWorldPosRadiansFeet,  
                          __in const P3DDXYZ &           vOrientationRadians,  
                          __in BOOL                  bOnGround,  
                          __out int &                 iEffectID  
)
```

private virtual

Turn effect on

Parameters

pszEffectName	Name of effect.
vWorldPosRadiansFeet	World position in Radians/Feet X=Lat Y=Alt Z=Lon.
vOrientationRadians	Orientation in Radians. X=Pitch Y=Heading Z=Bank.
bOnGround	Is effect on ground.

[out] **iEffectID** Effect id.

◆ GetEffectWorldOrientationRadians()

```
virtual HRESULT
```

	<pre>GetEffectWorldOrientationRadians (__in HANDLE hEffect, __out P3DDXYZ & vOrientationRadians)</pre> <p>Get effect world orientation in radians</p> <p>Parameters</p> <ul style="list-style-type: none"> hEffect Effect handle. vOrientationRadians Orientation in Radians. X=Pitch Y=Heading Z=Bank 	private virtual
◆ GetEffectWorldPosRadiansFeet()		
	<pre>virtual HRESULT GetEffectWorldPosRadiansFeet (__in HANDLE hEffect, __out P3DDXYZ & vWorldPosRadiansFeet)</pre> <p>Get effect world position in radians/feet</p> <p>Parameters</p> <ul style="list-style-type: none"> hEffect Effect handle. vWorldPosRadiansFeet World position in Radians/Feet X=Lat Y=Alt Z=Lon. 	private virtual
◆ RegisterVisualEffectCallback()		
	<pre>virtual HRESULT RegisterVisualEffectCallback (IVisualEffectCallbackV430 * pfCallback)</pre> <p>Registers a visual effect notification callback. The callback is issued when one or more visual effects are created.</p> <p>Parameters</p> <ul style="list-style-type: none"> pfCallback A pointer to the callback interface to be registered. <p>Returns</p> <ul style="list-style-type: none"> S_OK if the visual effect notification was successfully registered, E_FAIL otherwise. 	private pure virtual
◆ SetEffectDuration()		
	<pre>virtual HRESULT SetEffectDuration (__in int iEffectID, float fDurationSecs)</pre> <p>Set Effect Duration In Seconds</p> <p>Parameters</p> <ul style="list-style-type: none"> iEffectID Effect id. fDurationSecs Duration in seconds. 	private virtual
◆ SetEffectWorldOrientationRadians()		
	<pre>virtual HRESULT SetEffectWorldOrientationRadians (__in int iEffectID, __in const P3DDXYZ & vOrientationRadians)</pre> <p>Set effect world orientation in radians</p> <p>Parameters</p> <ul style="list-style-type: none"> iEffectID Effect id. vOrientationRadians Orientation in Radians. X=Pitch Y=Heading Z=Bank 	private virtual
◆ SetEffectWorldPosRadiansFeet()		
	<pre>virtual HRESULT SetEffectWorldPosRadiansFeet (__in int iEffectID, __in const P3DDXYZ & vWorldPosRadiansFeet)</pre>	private virtual

Set effect world position in radians/feet

Parameters

iEffectID Effect id.

vWorldPosRadiansFeet World position in Radians/Feet X=Lat Y=Alt Z=Lon.

◆ **UnRegisterVisualEffectCallback()**

virtual HRESULT

UnRegisterVisualEffectCallback (**IVisualEffectCallbackV430** * **pfCallback**) **private** **pure virtual**

Unregisters a visual effect notification callback.

Parameters

pfCallback A pointer to the callback interface to be unregistered.

Returns

S_OK if the visual effect notification was successfully unregistered, E_FAIL otherwise.

◆ **P3D::IVisualEffectCallbackV430**

class P3D::IVisualEffectCallbackV430

Inherits IUnknown.

Private Member Functions

virtual void **CreateEffect** (HANDLE hEffect, LPCWSTR pszEffectName, const **P3D::P3DDXYZ** &vLocation, const **P3D::P3DDXYZ** &vOrientation, UINT uObjectID) override

virtual void **DestroyEffect** (HANDLE hEffect) override

Member Function Documentation

◆ **CreateEffect()**

virtual void CreateEffect (HANDLE hEffect, LPCWSTR pszEffectName, const **P3D::P3DDXYZ** & vLocation, const **P3D::P3DDXYZ** & vOrientation, UINT uObjectID) **private** **pure virtual**

Called when a visual effect is created.

Parameters

hEffect Effect handle.

pszEffectName Name of effect.

vLocation Location of effect in radians/feet. If uObjectID != 0, it's the offset relative to the vehicle in feet.

vOrientation Orientation of effect in radians. If uObjectID != 0, it's the orientation relative to the vehicle in radians.

uObjectID Object's id, 0 if not attached.

◆ **DestroyEffect()**

virtual void DestroyEffect (HANDLE hEffect) **private** **pure virtual**

Called when a visual effect is destroyed.

Parameters

iEffectID Effect id.

◆ **P3D::IRopeSimulationV420**

class P3D::IRopeSimulationV420

This interface allows real-time updates of a rope created through the IVisualEffectManager.

Examples of uses would be helicopter hoists or tow plane cable.

Inherits IRopeSimulationV410.

Private Member Functions

```
virtual void SetStart (const P3DFXYZ &vStart) PURE
virtual P3DFXYZ GetStart () const PURE
virtual void SetEnd (const P3DFXYZ &vEnd) PURE
virtual P3DFXYZ GetEnd () const PURE
virtual void SetFixed (bool bStart, bool bVal) PURE
virtual bool GetFixed (bool bStart) const PURE
virtual void SetRelativeWind (const P3DFXYZ &vWind) PURE
virtual float GetRopeLength () const PURE
virtual HRESULT SetRopeLength (float fLength) PURE
virtual void SetRenderWorldPosition (const P3DDXYZ &vPos) PURE
virtual void SetRelativeGroundPosition (bool bCheckGround, float fAlt) PURE
virtual HRESULT SetRopeColor (in unsigned char uRed, in unsigned char uGreen, in
unsigned char uBlue, in unsigned char uAlpha) PURE
```

Member Function Documentation

◆ GetEnd()

```
virtual P3DFXYZ GetEnd ( ) const [private] [virtual]
```

Gets the rope ending offset, in feet, from the Lat/Lon/Alt specified in [SetRenderWorldPosition\(\)](#).

◆ GetFixed()

```
virtual bool GetFixed ( bool bStart ) const [private] [virtual]
```

Returns whether the rope start or end is "fixed".

Parameters

bStart Set to true if getting the starting point, false if the end point.

◆ GetRopeLength()

```
virtual float GetRopeLength ( ) const [private] [virtual]
```

Returns rope length, in feet

◆ GetStart()

```
virtual P3DFXYZ GetStart ( ) const [private] [virtual]
```

Gets the rope starting offset, in feet, from the Lat/Lon/Alt specified in [SetRenderWorldPosition\(\)](#).

◆ SetEnd()

```
virtual void SetEnd ( const P3DFXYZ & vEnd ) [private] [virtual]
```

Sets the rope ending offset. This is ignored if the ending point is fixed (see [SetFixed\(\)](#).)

Parameters

vEnd Offset, in feet, from the Lat/Lon/Alt specified in [SetRenderWorldPosition\(\)](#).

◆ SetFixed()

```
virtual void SetFixed ( bool bStart,
bool bVal )
```

```
)
```

```
private virtual
```

Sets whether the rope is fixed or not. "Fixed" means that the point will be set ([SetStart\(\)](#) or [SetEnd\(\)](#)) rather than allowing the rope simulation determine its position. For example, a helicopter hoist cable starting point would be "fixed" to the helicopter.

Parameters

bStart Set to true if changing the starting point, false if the end point.

vVal Set to true if "fixed", false otherwise.

◆ [SetRelativeGroundPosition\(\)](#)

```
virtual void SetRelativeGroundPosition ( bool bCheckGround,  
                                         float fAlt  
)
```

```
private virtual
```

Sets the relative ground position.

Parameters

bCheckGround Set to true if the simulation should include ground interaction.

fAlt The relative altitude, in feet, of the ground. This should be negative if the ground is below.

◆ [SetRelativeWind\(\)](#)

```
virtual void SetRelativeWind ( const P3DFXYZ & vWind )
```

```
private virtual
```

Sets the world-reference relative wind vector.

Parameters

vWind where X=longitude, Y=altitude, Z=latitude, in feet per second.

◆ [SetRenderWorldPosition\(\)](#)

```
virtual void SetRenderWorldPosition ( const P3DDXYZ & vPos )
```

```
private virtual
```

Sets the world position of the rope anchor point.

Parameters

vPos,where X=longitude (radians), Y=altitude (feet), Z=latitude (radians)

◆ [SetRopeColor\(\)](#)

```
virtual HRESULT SetRopeColor ( __in unsigned char uRed,  
                               __in unsigned char uGreen,  
                               __in unsigned char uBlue,  
                               __in unsigned char uAlpha  
)
```

```
private virtual
```

Sets the 8-bit color [RGBA](#) of the rope

Parameters

uRed Red channel code (0 - 255)

uGreen Green channel code (0 - 255)

uBlue Blue channel code (0 - 255)

uAlpha Alpha channel code (0 - 255)

◆ [SetRopeLength\(\)](#)

```
virtual HRESULT SetRopeLength ( float fLength )
```

```
private virtual
```

Sets new rope length

Parameters

fLength,in feet. The new rope length

◆ [SetStart\(\)](#)

```
virtual void SetStart( const P3DFXYZ & vStart ) [private] [virtual]
```

Sets the rope starting offset. This is ignored if the starting point is fixed (see [SetFixed\(\)](#).)

Parameters

vStart Offset, in feet, from the Lat/Lon/Alt specified in [SetRenderWorldPosition\(\)](#).

Variables

GUID **IID_IVisualEffectManagerV430**

VisualEffectManager interface ID. [More...](#)

GUID **SID_VisualEffectManager**

VisualEffectManager service ID. [More...](#)

GUID **IID_IVisualEffectCallbackV430**

GUID **IID_IRopeSimulationV420**

GUID **SID_RopeSimulation**

Service ID. [More...](#)

Variable Documentation

◆ **IID_IRopeSimulationV420**

GUID **IID_IRopeSimulationV420**

◆ **IID_IVisualEffectCallbackV430**

GUID **IID_IVisualEffectCallbackV430**

◆ **IID_IVisualEffectManagerV430**

GUID **IID_IVisualEffectManagerV430**

VisualEffectManager interface ID.

◆ **SID_RopeSimulation**

GUID **SID_RopeSimulation**

Service ID.

◆ **SID_VisualEffectManager**

GUID **SID_VisualEffectManager**

VisualEffectManager service ID.

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Window and Camera Services

Overview

The window and camera services enables external applications to control basic camera functions as well as adding/removing post-process effects and sensor modes using window plug-ins.

Classes

- struct **PickResult**
- class **IMouseRectListenerCallback**
- class **IWindowPluginSystemV440**
- class **ICameraSystemV451**
- class **IWindowV440**
- class **WindowPlugin**

Class Documentation

◆ P3D::PickResult

struct P3D::PickResult

Public Member Functions

[PickResult \(\) noexcept](#)

Public Attributes

bool **bWasHit**

ScreenCoord **Location**

unsigned int **uMessage**

float **fTextureU**

float **fTextureV**

IWindowV400 * **pWindow**

Constructor & Destructor Documentation

◆ PickResult()

[PickResult \(\) inline noexcept](#)

Member Data Documentation

◆ bWasHit

bool bWasHit

◆ fTextureU

float fTextureU

◆ fTextureV

float fTextureV

◆ Location

ScreenCoord Location

◆ pWindow

IWindowV400* pWindow

◆ uMessage

unsigned int uMessage

◆ P3D::IMouseRectListenerCallback

class P3D::IMouseRectListenerCallback

Interface used for registering and unregistering for mouse rectangle click event callbacks. Below is an example implementation of this interface:

```
class MyMouseRectListenerCallback : public
    P3D::IMouseRectListenerCallback
{
    DEFAULT_REFCOUNT_INLINE_IMPL();

    STDMETHODIMP QueryInterface(REFIID riid, PVOID* ppv)
    {
        HRESULT hr = E_NOINTERFACE;

        if (ppv == NULL)
        {
            return E_POINTER;
        }

        *ppv = NULL;

        if (IsEqualIID(riid, IID_IMouseListenerCallback))
        {
            *ppv = static_cast(this);
        }
        else if (IsEqualIID(riid, IID_IUnknown))
        {
            *ppv = static_cast(this);
        }
        if (*ppv)
        {
            hr = S_OK;
            AddRef();
        }

        return hr;
    }

public:
    MyMouseRectListenerCallback() :
        m_RefCount(1)
    {}

    virtual void MouseRectListenerProc(UINT id, MOUSE_CLICK_TYPE
        clickType) override;
};

// override
void MyMouseRectListenerCallback::MouseRectListenerProc(UINT id,
    MOUSE_CLICK_TYPE clickType)
{
    OutputDebugString(_T("Click received!"));
}
```

Inherits IUnknown.

Private Member Functions

virtual void **MouseRectListenerProc** (UINT id, MOUSE_CLICK_TYPE clickType) override

Member Function Documentation

◆ MouseRectListenerProc()

```
virtual void MouseRectListenerProc( UINT           id,
                                    MOUSE_CLICK_TYPE clickType
                                )
```

private pure virtual

This function is called when a mouse rectangle is clicked.

Parameters

id id of the mouse rect hit
clickType click type

◆ P3D::IWindowPluginSystemV440

class P3D::IWindowPluginSystemV440

Service for accessing windows and registering window plugins.

Inherits IWindowPluginSystemV420.

Private Member Functions

```
virtual HRESULT RegisterMouseRectListenerCallback (IMouseRectListenerCallback *callback)
override
virtual HRESULT UnRegisterMouseRectListenerCallback (IMouseRectListenerCallback
*callback) override
virtual HRESULT FireMouseRectClick (UINT id, MOUSE_CLICK_TYPE clickType) override
virtual BOOL HasWindow (LPCWSTR name) override
virtual HRESULT GetWindowList (IWindowList &windows) override
virtual HRESULT GetWindow (LPCWSTR name, IWindowV400 *&window) override
virtual HRESULT RegisterInternalWindow (LPCWSTR name, IWindowPluginV400 *plugin) override
virtual HRESULT UnRegisterInternalWindow (LPCWSTR name, IWindowPluginV400 *callback)
override
virtual HRESULT CreateInternalWindow (LPCWSTR name, UINT32 x, UINT32 y, UINT32 width,
UINT32 height, IWindowPluginV400 *pPlugin, bool bRTT=false) override
virtual HRESULT CloseInternalWindow (LPCWSTR name) override
virtual IWindowV400 * GetCurrentWindow () override
virtual HRESULT CreateCameraInstance (const GUID &guidCameraDefinition, LPCWSTR
pszName, UINT uObjectId, UINT &uInstanceld) override
virtual HRESULT CreateCameraInstance (const GUID &guidCameraDefinition, LPCWSTR
pszName, double dLatDegrees, double dLonDegrees, double dAltMeters, UINT
&uInstanceld) override
virtual HRESULT DeleteCameraInstance (const GUID &guidCameraDefinition, UINT uInstanceld)
override
virtual IWindowV400 * GetDefaultWindow () override
```

Member Function Documentation

◆ CloseInternalWindow()

virtual HRESULT CloseInternalWindow (LPCWSTR name) [private] [pure virtual]

Close an existing window Return error if window does not exist

Parameters

[in] **name** Name of the window

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ CreateCameraInstance() [1/2]

```
virtual HRESULT CreateCameraInstance ( const GUID & guidCameraDefinition,
LPCWSTR pszName,
UINT uObjectId,
UINT & uInstanceld
)
```

[private] [pure virtual]

This function is used to create a new camera instance on the given object based on the given camera definition.

Parameters

[in] **guidCameraDefinition** The guid of the camera definition to be used to create the camera.
[in] **pszName** The name of the camera.
[in] **uObjectId** The id of the object that this camera instance will be associated with.
[out] **uInstanceld** The instance id of the created camera.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails.

◆ CreateCameraInstance() [2/2]

virtual HRESULT CreateCameraInstance (const GUID & guidCameraDefinition,

```

LPCWSTR pszName,
double dLatDegrees,
double dLonDegrees,
double dAltMeters,
UINT & ulInstanceId
)

```

private pure virtual

This function is used to create a new camera instance at the given location based on the given camera definition.

Parameters

[in] guidCameraDefinition	The guid of the camera definition to be used to create the camera.
[in] pszName	The name of the camera.
[in] dLatDegrees	The latitude of the camera in degrees.
[in] dLonDegrees	The longitude of the camera in degrees.
[in] dAltMeters	The altitude of the camera in meters.
[out] ulInstanceId	The instance id of the created camera.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails.

◆ CreateInternalWindow()

```

virtual HRESULT CreateInternalWindow ( LPCWSTR name,
                                     UINT32 x,
                                     UINT32 y,
                                     UINT32 width,
                                     UINT32 height,
                                     IWindowPluginV400 * pPlugin,
                                     bool bRTT = false
)

```

private pure virtual

Create a new window and register a callback. Return error if window already exist

Parameters

[in] name , Name	of the window
[in] x,y	Window position relative to parent window. (origin is top left corner)
[in] width,height	Window dimensions in pixels
[in] plugin	Plugin that will be registered at creation time
[in] bRTT	Determines if this window is a render-to-texture view.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ DeleteCameraInstance()

```

virtual HRESULT DeleteCameraInstance ( const GUID & guidCameraDefinition,
                                      UINT ulInstanceId
)

```

private pure virtual

This function is used to delete a camera instance.

Parameters

[in] guidCameraDefinition	The guid of the camera definition to be used to delete the camera instance.
[in] ulInstanceId	The instance id of the camera to be deleted.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails.

◆ FireMouseRectClick()

```

virtual HRESULT FireMouseRectClick ( UINT id,
                                    MOUSE_CLICK_TYPE clickType
)

```

private pure virtual

Used to fire a mouse rectangle with the given ID and click type.

Parameters

id	id of the mouse rect hit
clickType	click type

◆ GetCurrentWindow()

```
virtual IWindowV400* GetCurrentWindow( ) [private] [pure virtual]
```

This will return the window currently being rendered or updated. If no windows are being rendered, or updated, this will return the active 3d window. If no 3d view windows are active, this will return the default 3d view.

Returns

Current window pointer. Will return nullptr if it request fails.

◆ **GetDefaultWindow()**

```
virtual IWindowV400* GetDefaultWindow( ) [private] [pure virtual]
```

This will return the default window.

Returns

default window pointer. Will return nullptr if it request fails.

◆ **GetWindow()**

```
virtual HRESULT GetWindow( LPCWSTR name,  
                           IWindowV400 * & window  
                         ) [private] [pure virtual]
```

Get a IWindowReaderPdk interface for a 3D view window with the input name

Parameters

[in] **name** Name of the window

[out] **window** The window with the requested name

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ **GetWindowList()**

```
virtual HRESULT GetWindowList( IWindowList & windows ) [private] [pure virtual]
```

Get list of windows containing a 3D view that are currently open

Parameters

[out] **windows** list of windows

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ **HasWindow()**

```
virtual BOOL HasWindow( LPCWSTR name ) [private] [pure virtual]
```

Check if a window containing a 3d view exists

Returns

name Name of the window

◆ **RegisterInternalWindow()**

```
virtual HRESULT RegisterInternalWindow( LPCWSTR name,  
                                       IWindowPluginV400 * plugin  
                                     ) [private] [pure virtual]
```

Register a callback to an existing window. Return error if window does not exist

Parameters

[in] **name** Name of the window

[in] **plugin** Plugin that will be registered to the window

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

Remarks

While the name is used to find the window, the plugin is added to the window and will continue to get called even if the camera changes or the window name changes.

◆ **RegisterMouseRectListenerCallback()**

```
virtual HRESULT  
RegisterMouseRectListenerCallback ( IMouseRectListenerCallback * callback ) [private] [pure virtual]
```

Register a callback to be hit when a mouse rectangle is clicked.

Parameters

callback The callback to register

◆ **UnRegisterInternalWindow()**

```
virtual HRESULT UnRegisterInternalWindow ( LPCWSTR name,  
                                         IWindowPluginV400 * callback  
                                         ) [private] [pure virtual]
```

Unregister a callback from an existing window. Return error if window does not exist

Parameters

[in] **name** Name of the window
[in] **plugin** Plugin that will be unregistered

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ **UnRegisterMouseRectListenerCallback()**

```
virtual HRESULT  
UnRegisterMouseRectListenerCallback ( IMouseRectListenerCallback * callback ) [private] [pure virtual]
```

UnRegister a callback that would be hit when a mouse rectangle is clicked.

Parameters

callback The callback to unregister

◆ P3D::ICameraSystemV451

Camera system interface that can be used to get or set camera states such as position, orientation, and field of view.

Remarks

The reader/writer interfaces usind in Prepar3D v2 and v3 have been have been combined for ease of use and the camera and window functionality have been put in separate interfaces. All 3d views will have a camera system. Panel windows do not have a camera system.

Inherits ICameraSystemV450.

Private Member Functions

```
virtual void GetCameraDefinition (WCHAR *cameraDefinition)  
override
```

```
virtual void SetRelative6DOF (float fDeltaX, float fDeltaY, float  
fDeltaZ, float fPitchDeg, float fBankDeg, float  
fHeadingDeg) override
```

```
virtual void SetSmoothRelative6DOF (float fDeltaX, float fDeltaY,  
float fDeltaZ, float fPitchDeg, float fBankDeg, float  
fHeadingDeg, float smoothPanTime=0) override
```

```
virtual void SetFov (float hFov, float vFov) override
```

```
virtual void SetZoomGoal (float zoomGoal) override
```

```
virtual void SetZoom (float zoom) override
```

```
virtual void PanToView (const WCHAR *name) override
```

```
virtual void AddPostProcess (const WCHAR *name) override
```

```
virtual void RemovePostProcess (const WCHAR *name) override
```

```
virtual void ResetPostProcess () override
```

```
virtual void ClearPostProcess () override
```

```
virtual void Reset () override
```

```
virtual void ZoomIn (void) override
```

```
virtual void ZoomOut (void) override
```

```
virtual void SetVirtualCockpitTransparentValue (unsigned int level)  
override
```

```
virtual void SetChaseDistance (float fDistance) override
```

```
virtual void SetChaseAltitude (float fAltitude) override
```

```
virtual void SetSensorMode (unsigned int mode) override
```

```
virtual void SetExcludeVcPostProcess (bool exclude) override
```

virtual void	SetExcludeVcPanelsSensor (bool exclude) override
virtual void	SetFarClip (float far) override
virtual void	SetNearClip (float near) override
virtual void	SetTerrainLODOriginLLA (double lat, double lon, double alt) override
virtual void	SetSceneryLODOriginLLA (double lat, double lon, double alt) override
virtual void	SetAmbientBoostAndLightAmplificationLevel (const float ambientBoost, const float lightAmplificationLevel) override
virtual void	SetTargetFrameRate (const float targetFrameRate) override
virtual int	AddPickRequest (int x, int y) override
virtual void	ActivatePositionTracking () override
virtual void	DeactivatePositionTracking () override
virtual void	ActivateEntityTracking () override
virtual void	DeactivateEntity Tracking () override
virtual void	SetTargetLatLonAltDegrees (double lat, double lon, double alt) override
virtual void	SetTargetContainerId (UINT containerId) override
virtual void	SetPBH (float fPitchDeg, float fBankDeg, float fHeadingDeg) override
virtual void	SetLLA (double dLat, double dLon, double dAlt) override
virtual void	SetOffsetXYZ (float fDeltaX, float fDeltaY, float fDeltaZ) override
virtual void	SetGlobalRotate (bool bGlobalRotate) override
virtual void	SetGlobalPBH (float fPitchDeg, float fBankDeg, float fHeadingDeg) override
virtual void	TargetCameraLookAt () override
virtual void	SetSideAngles (float fLeft, float fRight, float fTop, float fBottom) override
virtual float	GetRightAngle () override
virtual float	GetLeftAngle () override
virtual float	GetBottomAngle () override
virtual float	GetTopAngle () override
virtual void	SetFrustumOffsetXYZ (float fOffsetX, float fOffsetY, float fOffsetZ) override
virtual void	SetFrustumOffsetPBH (float fOffsetP, float fOffsetB, float fOffsetH) override
virtual void	SetHmdMode (HMD_MODE eMode) override
virtual int	GetActiveViewGroup () override
virtual void	GetFov (float &fFov, float &vFov) const override
virtual void	GetZoom (float &fZoom) const override
virtual void	GetPBH (float &p, float &b, float &h) const override
virtual void	GetBiasPBH (float &p, float &b, float &h) const override
virtual void	GetLLA (double &lat, double &lon, double &alt) const override
virtual void	GetCameraOffset (float &x, float &y, float &z) const override
virtual void	GetViewMatrix (float output4x4[4][4]) const override
virtual void	GetStandardProjectionMatrix (float output4x4[4][4]) const override
virtual void	GetVirtualCockpitProjectionMatrix (float output4x4[4][4]) const override
virtual float	GetNearClip () const override
virtual float	GetFarClip () const override
virtual void	GetPickResult (int ID, bool &wasHit, float &distanceToHit) const override
virtual void	GetPickResult (int ID, PickResult &result) const override
virtual int	GetAvatarMode () const override
virtual void	GetLLARadians (double &lat, double &lon, double &alt) const override
virtual void	GetPBHRadians (float &pitch, float &bank, float &heading) const override
virtual void	GetTargetLatLonAltDegrees (double &lat, double &lon, double &alt) const override

virtual void	GetTargetContainerId (UINT32 &containerId) const override
virtual void	GetSensorMode (UINT32 &sensorMode) const override
virtual bool	GetCameraLookAtLLA (double &lat, double &lon, double &alt) const override
virtual bool	IsGlobalRotate () const override
virtual void	GetGlobalPBH (float &p, float &b, float &h) const override
virtual void	GetTargetLatString (WCHAR *targetLat) const override
virtual void	GetTargetLonString (WCHAR *targetLon) const override
virtual void	GetPickTextureLocation (int ID, WCHAR *windowName, bool &wasHit, float &u, float &v, float &distanceToHit) const override
virtual void	GetScreenCoord (const double lla[3], const float pbh[3], const float xyzOffset[][3], float xyOutput[][3], const int count) const override
virtual void	GetScreenCoord (const float xyzOffset[][3], float xyOutput[][3], const int count) const override
virtual void	GetScreenCoord (const LLADegreesMeters &lla, ScreenCoord &screenCoord) const override
virtual void	GetWorldCoord (const ScreenCoord &screenCoord, LLADegreesMeters &lla) const override
virtual unsigned int	GetVirtualCockpitTransparentValue () const override
virtual float	GetChaseDistance () const override
virtual float	GetChaseAltitude () const override
virtual float	GetTargetFrameRate () const override
virtual void	GetPostProcesses (INameList &names) const override
virtual bool	IsVirtualCockpit (void) const override
virtual bool	IsTopDown (void) const override
virtual void	GetRequestedZoom (float &zoom) const override
virtual bool	IsHMDView () const override
virtual void	GetWorldTransform (ObjectWorldTransform &transform, CAMERA_TRANFORM_REFERENCE referece) const override
virtual void	GetSideAngles (float &fLeft, float &fRight, float &fTop, float &fBottom) const override
virtual bool	GetTargetCrosshairs () const override
virtual bool	GetRenderDesignators () const override
virtual LPCWSTR	GetToolTipText () const override
virtual float	GetGroundAlt () override
virtual void	SetUseGlobalTerrainView (bool useGlobalTerrainView) override
virtual void	SetSinglePassGroupID (unsigned int id, unsigned int drawOrder) override
virtual unsigned int	GetSinglePassGroupID () const override
virtual CAMERA_SYSTEM_ORIGIN	GetCameraOrigin () const override
virtual CAMERA_SYSTEM_CATEGORY	GetCameraCategory () const override
virtual bool	MouseGetCursorIsHidden () override
virtual void	ResetMouseCursorCountdown () override
virtual LPCWSTR	GetToolTipText (int ePickID) const override

Member Function Documentation

◆ ActivateEntityTracking()

virtual void ActivateEntityTracking () private pure virtual

Activate entity tracking enabling camera to look at a specified entity

◆ ActivatePositionTracking()

virtual void ActivatePositionTracking () private pure virtual

Activate position tracking enabling camera to look at a specified LLA

◆ AddPickRequest()

```
virtual int AddPickRequest ( int x,  
                           int y  
                         ) [private] [pure virtual]
```

Request a Pick test at a given screen coordinate. An ID is returned which can later be used to request the results of the test.

Parameters

x screen coordinate
y screen coordinate

Returns

ID later used to request the pick result

◆ **AddPostProcess()**

```
virtual void AddPostProcess ( const WCHAR * name ) [private] [pure virtual]
```

Add a post process effect to the window

Parameters

name Name of post process to add

◆ **ClearPostProcess()**

```
virtual void ClearPostProcess ( ) [private] [pure virtual]
```

Clear all post process effects from the camera

◆ **DeactivateEntityTracking()**

```
virtual void DeactivateEntityTracking ( ) [private] [pure virtual]
```

Deactivate entity tracking

◆ **DeactivatePositionTracking()**

```
virtual void DeactivatePositionTracking ( ) [private] [pure virtual]
```

Deactivate position tracking

◆ **GetActiveViewGroup()**

```
virtual int GetActiveViewGroup ( ) [private] [pure virtual]
```

Get active View Group

◆ **GetAvatarMode()**

```
virtual int GetAvatarMode ( ) const [private] [pure virtual]
```

Get the AvatarMode for the current window.

Returns

The AvatarMode of the current window. (0 = None, 1 = First, 2 = Third)

◆ **GetBiasPBH()**

```
virtual void GetBiasPBH ( float & p,  
                           float & b,  
                           float & h  
                         ) const [private] [pure virtual]
```

Get the offset of the pbh relative to the origin

Parameters

[out] **p** Pitch
[out] **b** Bank
[out] **h** Heading

◆ **GetBottomAngle()**

virtual float GetBottomAngle() **private** **pure virtual**

Gets the Bottom Side Angle of the camera view

◆ **GetCameraCategory()**

virtual **CAMERA_SYSTEM_CATEGORY** GetCameraCategory() const **private** **pure virtual**

◆ **GetCameraDefinition()**

virtual void GetCameraDefinition (**WCHAR** * **cameraDefinition**) **private** **pure virtual**

Get the window's camera definition

Remarks

camera definitions are defined in Cameras.cfg or in aircraft.cfg

◆ **GetCameraLookAtLLA()**

virtual bool GetCameraLookAtLLA (double & **lat**,
double & **lon**,
double & **alt**
) **const** **private** **pure virtual**

Gets the LLA the camera is looking at

Parameters

[out] **lat** The target latitude
[out] **lon** The target longitude
[out] **alt** The target altitude

Returns

If the LLA is valid

◆ **GetCameraOffset()**

virtual void GetCameraOffset (float & **x**,
float & **y**,
float & **z**
) **const** **private** **pure virtual**

Get the camera offset from the origin

Parameters

[out] **x,y,z** camera's 3d offset from origin

◆ **GetCameraOrigin()**

virtual **CAMERA_SYSTEM_ORIGIN** GetCameraOrigin() const **private** **pure virtual**

◆ **GetChaseAltitude()**

virtual float GetChaseAltitude() const **private** **pure virtual**

Gets the chase altitude in meters

Returns

Chase altitude in meters

◆ **GetChaseDistance()**

virtual float GetChaseDistance() const **private** **pure virtual**

Gets the chase distance in meters

Returns

Chase distance in meters

◆ GetFarClip()

```
virtual float GetFarClip( ) const [private] [pure virtual]
```

Gets far clip distance in meters

Returns

far clip distance in meters

◆ GetFov()

```
virtual void GetFov( float & hFov,  
                     float & vFov  
                   ) const [private] [pure virtual]
```

Get the field of view of the camera

Parameters

[out] **hFov** Horizontal field of view
[out] **vFov** Vertical field of view

◆ GetGlobalPBH()

```
virtual void GetGlobalPBH( float & p,  
                          float & b,  
                          float & h  
                        ) const [private] [pure virtual]
```

Gets the global rotation PBH of the camera

Parameters

[out] **p** The pitch in degrees
[out] **b** The bank in degrees
[out] **h** The heading in degrees

◆ GetGroundAlt()

```
virtual float GetGroundAlt( ) [private] [pure virtual]
```

◆ GetLeftAngle()

```
virtual float GetLeftAngle( ) [private] [pure virtual]
```

Gets the Left Side Angle of the camera view

◆ GetLLA()

```
virtual void GetLLA( double & lat,  
                     double & lon,  
                     double & alt  
                   ) const [private] [pure virtual]
```

Get the offset of the latitude longitude and altitude of the camera

Parameters

[out] **lat** Latitude
[out] **lon** Longitude
[out] **alt** Altitude

◆ GetLLARadians()

```
virtual void GetLLARadians( double & lat,  
                            double & lon,  
                            double & alt  
                          ) const [private] [pure virtual]
```

Get the offset of the latitude longitude and altitude of the camera in radians

Parameters

[out] **lat** Latitude
[out] **lon** Longitude
[out] **alt** Altitude

◆ GetNearClip()

```
virtual float GetNearClip( ) const [private] [pure virtual]
```

Gets near clip distance in meters

Returns

near clip distance in meters

◆ GetPBH()

```
virtual void GetPBH ( float & p,  
                      float & b,  
                      float & h  
                    ) const [private] [pure virtual]
```

Get the orientation of the camera (Pitch, Bank, and Heading)

Parameters

[out] **p** Pitch
[out] **b** Bank
[out] **h** Heading

◆ GetPBHRadians()

```
virtual void GetPBHRadians ( float & pitch,  
                            float & bank,  
                            float & heading  
                          ) const [private] [pure virtual]
```

Get the orientation of the camera (Pitch, Bank, and Heading) in radians

Parameters

[out] **p** Pitch
[out] **b** Bank
[out] **h** Heading

◆ GetPickResult() [1/2]

```
virtual void GetPickResult ( int ID,  
                           bool & wasHit,  
                           float & distanceToHit  
                         ) const [private] [pure virtual]
```

Get hit result. A pick request must first be requested using AddPickRequest. The ID passed in should be the ID that was returned from the AddPickRequest call. Passing in 0 will return the last result from the native mouse handling.

Parameters

ID ID of the hit request.
[out] **wasHit** true if hit test passed
[out] **distance** distance to hit test intersection

◆ GetPickResult() [2/2]

```
virtual void GetPickResult ( int ID,  
                           PickResult & result  
                         ) const [private] [pure virtual]
```

Get hit result. A pick request must first be requested using AddPickRequest. The ID passed in should be the ID that was returned from the AddPickRequest call. Passing in 0 will return the last result from the native mouse handling.

Parameters

ID ID of the hit request.
[out] **result** The hit result

◆ GetPickTextureLocation()

```
virtual void GetPickTextureLocation ( int ID,
```

```

WCHAR * windowName,
bool & wasHit,
float & u,
float & v,
float & distanceToHit
) const private pure virtual

```

Get the texture coords that the pick hit on the texture in the specified window. A pick request must first be requested using AddPickRequest on a WindowPdk interface from within a window callback. The ID passed in should be the ID that was returned from the AddPickRequest call.

Parameters

ID	ID of the hit request.
windowName	Name of window texture resides within.
[out] wasHit	true if hit test passed
[out] u	u location in texture
[out] v	v location in texture
[out] distanceToHit	distance to texture hit

◆ GetPostProcesses()

```
virtual void GetPostProcesses ( INameList & names ) const private pure virtual
```

Gets list of the names of post processes assigned to this window

Parameters

[out] nameList	array of cstrings to hold the list of names (size should be 32)
[out] size	of list returned (will not be larger than 32)

◆ GetRenderDesignators()

```
virtual bool GetRenderDesignators ( ) const private pure virtual
```

Gets whether or not the camera has a designator

Returns

true if camera has designator

◆ GetRequestedZoom()

```
virtual void GetRequestedZoom( float & zoom ) const private pure virtual
```

Get the requested zoom from the camera

Parameters

[out] the	requested zoom of the camera
------------------	------------------------------

◆ GetRightAngle()

```
virtual float GetRightAngle ( ) private pure virtual
```

Gets the Right Side Angle of the camera view

◆ GetScreenCoord() [1/3]

```

virtual void GetScreenCoord ( const double lla[3],
                           const float pbh[3],
                           const float xyzOffset[][3],
                           float xyOutput[][3],
                           const int count
) const private pure virtual

```

Get the screen coord of a point LLA + xyz offset in world space (meters)

Parameters

[in] lla	Lat lon alt origin of local coordinate system
[in] pbh	Orientation of local coordinate system
[in] xyzOffset	List of 3d offsets in local coordinates.
[out] xyOutput	List of screenspace position of the requested points
[in] count	Number of points to request. Must be at least as big as input array sizes.

◆ GetScreenCoord() [2/3]

```
virtual void GetScreenCoord ( const float xyzOffset[][3],  
                           float xyOutput[][3],  
                           const int count  
                         ) const private pure virtual
```

Get the screen space coordinates of a list of points relative to user sim object

Parameters

[in] **xyzOffset** list of 3d offsets in local coordinates.
[out] **xyOutput** list of screenspace position of the requested points
[in] **count** Number of points to request. Must be at least as big as input array sizes.

◆ GetScreenCoord() [3/3]

```
virtual void GetScreenCoord ( const LLADegreesMeters & lla,  
                           ScreenCoord & screenCoord  
                         ) const private pure virtual
```

Get the screen space coordinates of a point at the given LLA

Parameters

[in] **lla** world point to query
[out] **screenCoord** ScreenCoord of output

◆ GetSensorMode()

```
virtual void GetSensorMode ( UINT32 & sensorMode ) const private pure virtual
```

Get the sensor mode of the camera.

Parameters

[out] **sensorMode** The sensor mode of the camera

◆ GetSideAngles()

```
virtual void GetSideAngles ( float & fLeft,  
                           float & fRight,  
                           float & fTop,  
                           float & fBottom  
                         ) const private pure virtual
```

Get the side angles of the camera view in degrees

Parameters

[out] **fLeft** left side angle
[out] **fRight** right side angle
[out] **fTop** top side angle
[out] **fBottom** bottom side angle

◆ GetSinglePassGroupID()

```
virtual unsigned int GetSinglePassGroupID ( ) const private pure virtual
```

Gets the single pass group id for the camera view

Returns

current single pass group ID, 0 means not assigned to a group

◆ GetStandardProjectionMatrix()

```
virtual void GetStandardProjectionMatrix ( float output4x4[4][4] ) const private pure virtual
```

Get the standard projection matrix which is used by the Camera/Window for rendering everything but the virtual cockpit

Parameters

[out] **output4x4** 4x4 vidw matrix

◆ GetTargetContainerId()

```
virtual void GetTargetContainerId ( UINT32 & containerId ) const private pure virtual
```

Get the target container ID for the camera to track.

Parameters

[out] **containerId** The ID of the container to track

◆ **GetTargetCrosshairs()**

```
virtual bool GetTargetCrosshairs ( ) const private pure virtual
```

Gets whether or not the camera has crosshairs

Returns

true if camera is an HDM view

◆ **GetTargetFrameRate()**

```
virtual float GetTargetFrameRate ( ) const private pure virtual
```

Gets the target frame rate in seconds

Returns

Target frame rate in seconds

◆ **GetTargetLatLonAltDegrees()**

```
virtual void GetTargetLatLonAltDegrees ( double & lat,  
                                         double & lon,  
                                         double & alt  
                                       ) const private pure virtual
```

Get the target location for the camera to track.

Parameters

[out] **lat** Target latitude in degrees

[out] **lon** Target longitude in degrees

[out] **alt** Target altitude in feet

◆ **GetTargetLatString()**

```
virtual void GetTargetLatString ( WCHAR * targetLat ) const private pure virtual
```

Gets the target latitude in string format

Parameters

[out] **targetLat** Latitude

◆ **GetTargetLonString()**

```
virtual void GetTargetLonString ( WCHAR * targetLon ) const private pure virtual
```

Gets the target longitude in string format

Parameters

[out] **targetLon** Longitude

◆ **GetToolTipText() [1/2]**

```
virtual LPCWSTR GetToolTipText ( ) const private pure virtual
```

◆ **GetToolTipText() [2/2]**

```
virtual LPCWSTR GetToolTipText ( int ePickID ) const private pure virtual
```

Returns the tooltip for an element intersected by a specific external pick request, based on pick ID

Returns

L "" if no valid tooltip, otherwise the tooltip of the element hit by the external pick request

Gets if the camera is rotating relative to the world or vehicle.

Returns

Whether or not the camera is globally rotating

◆ **IsHMDView()**

virtual bool IsHMDView () const **private** **pure virtual**

Check if camera is an HMD view

Returns

true if camera is an HMD view

◆ **IsTopDown()**

virtual bool IsTopDown (void) const **private** **pure virtual**

Check if camera is Top Down view

Returns

true if window's active camera is a Top Down view

◆ **IsVirtualCockpit()**

virtual bool IsVirtualCockpit (void) const **private** **pure virtual**

Check if camera is a virtual cockpit

Returns

true if window's active camera is a virtual cockpit view

◆ **MouseGetCursorIsHidden()**

virtual bool MouseGetCursorIsHidden () **private** **pure virtual**

Gets whether the mouse cursor is currently hidden

Returns

true if the mouse cursor is hidden, false if the mouse cursor is being shown

◆ **PanToView()**

virtual void PanToView (const WCHAR * **name**) **private** **pure virtual**

Pan the camera to a saved location

Parameters

name of saved custom camera to pan to

◆ **RemovePostProcess()**

virtual void RemovePostProcess (const WCHAR * **name**) **private** **pure virtual**

Remove a post process effect to the window

Parameters

name Name of post process to remove

◆ **Reset()**

virtual void Reset () **private** **pure virtual**

Reset camera settings to those defined in camera definition

◆ **ResetMouseCursorCountdown()**

virtual void ResetMouseCursorCountdown () **private** **pure virtual**

Resets the timer to hide the mouse cursor

◆ **ResetPostProcess()**

virtual void ResetPostProcess () **private** **pure virtual**

Reset all post process effects to those defined in camera definition

◆ **SetAmbientBoostAndLightAmplificationLevel()**

virtual void SetAmbientBoostAndLightAmplificationLevel (const float **ambientBoost**,
const float **lightAmplificationLevel**
)

private **pure virtual**

Set the a scalar to amplify all light, and or a boost to the base ambient level. defaults are 1.0 and 0.0

Parameters

ambientBoost boost to base ambient level
lightAmplificationLevel scalar multiplied into all light sources

◆ **SetChaseAltitude()**

virtual void SetChaseAltitude (float **fAltitude**) **private** **pure virtual**

Sets the chase altitude of a camera

Parameters

fAltitude Chase altitude in meters

◆ **SetChaseDistance()**

virtual void SetChaseDistance (float **fDistance**) **private** **pure virtual**

Sets the chase distance of a camera

Parameters

fDistance Chase distance in meters

◆ **SetExcludeVcPanelsSensor()**

virtual void SetExcludeVcPanelsSensor (bool **exclude**) **private** **pure virtual**

Excludes VC panels from being sensorized

Parameters

exclude boolean value to set exclusion state

◆ **SetExcludeVcPostProcess()**

virtual void SetExcludeVcPostProcess (bool **exclude**) **private** **pure virtual**

Excludes entire VC from being post processed (if post process has PreVC enabled)

Parameters

exclude boolean value to set exclusion state

◆ **SetFarClip()**

virtual void SetFarClip (float **far**) **private** **pure virtual**

Set the far clip.

Parameters

far Far clip distance in meters

◆ **SetFov()**

virtual void SetFov (float **hFov**,
float **vFov**
)

private **pure virtual**

Set the field of view of the camera

Parameters

hFov Horizontal field of view
vFov Vertical field of view

◆ **SetFrustumOffsetPBH()**

```
virtual void SetFrustumOffsetPBH ( float fOffsetP,  
                                 float fOffsetB,  
                                 float fOffsetH  
                               )
```

[private] [pure virtual]

Set the offset PBH rotation relative to frustum group parent rotation

Parameters

fOffsetP Offset frustum Pitch relative to parent of group
fOffsetB Offset frustum Bank relative to parent of group
fOffsetH Offset frustum Heading relative to parent of group

◆ **SetFrustumOffsetXYZ()**

```
virtual void SetFrustumOffsetXYZ ( float fOffsetX,  
                                 float fOffsetY,  
                                 float fOffsetZ  
                               )
```

[private] [pure virtual]

Set the offset from the camera group origin for this camera

Parameters

fOffsetX Offset frustum from group origin in x direction relative to origin
fOffsetY Offset frustum from group origin in y direction relative to origin
fOffsetZ Offset frustum from group origin in z direction relative to origin

◆ **SetGlobalPBH()**

```
virtual void SetGlobalPBH ( float fPitchDeg,  
                           float fBankDeg,  
                           float fHeadingDeg  
                         )
```

[private] [pure virtual]

Sets the global rotation value of the camera

Parameters

fPitchDeg The pitch
fBankDeg The bank
fHeadingDeg The heading

◆ **SetGlobalRotate()**

```
virtual void SetGlobalRotate ( bool bGlobalRotate )
```

[private] [pure virtual]

Sets whether the camera will rotate independent of attached object

Parameters

bGlobalRotate Whether or not the camera will rotate independent of attached object

◆ **SetHmdMode()**

```
virtual void SetHmdMode ( HMD_MODE eMode )
```

[private] [pure virtual]

Set the HMD mode for this camera

Parameters

eMode the HDM mode to assign to this camera

◆ **SetLLA()**

```
virtual void SetLLA ( double dLat,  
                     double dLon,
```

```
    double dAlt  
)  
) private pure virtual
```

Set the location of the camera. Currently only applies to cameras with a Fixed Origin type.

Parameters

dLat Latitude in degrees
dLon Longitude in degrees
dAlt Altitude in meters

◆ **SetNearClip()**

```
virtual void SetNearClip ( float near ) private pure virtual
```

Set the near clip.

Parameters

near Near clip distance in meters

◆ **SetOffsetXYZ()**

```
virtual void SetOffsetXYZ ( float fDeltaX,  
                           float fDeltaY,  
                           float fDeltaZ  
) private pure virtual
```

Set the xyz offset of the camera.

Parameters

fDeltaX X offset in meters
fDeltaY Y offset in meters
fDeltaZ Z offset in meters

◆ **SetPBH()**

```
virtual void SetPBH ( float fPitchDeg,  
                      float fBankDeg,  
                      float fHeadingDeg  
) private pure virtual
```

Set the pbh of the camera.

Parameters

fPitchDeg Pitch in degrees
fPitchDeg Bank, in degrees
fPitchDeg Heading in degrees

◆ **SetRelative6DOF()**

```
virtual void SetRelative6DOF ( float fDeltaX,  
                             float fDeltaY,  
                             float fDeltaZ,  
                             float fPitchDeg,  
                             float fBankDeg,  
                             float fHeadingDeg  
) private pure virtual
```

Set the camera position and orientation relative to its origin in 3D space

Parameters

fDeltaX,fDeltaY,fDeltaZ 3D position relative to origin
fPitchDeg,fBankDeg,fHeadingDeg Relative orientation pitch bank and heading

◆ **SetSceneryLODOriginLLA()**

```
virtual void SetSceneryLODOriginLLA ( double lat,  
                                       double lon,  
                                       double alt  
) private pure virtual
```

Set the LLA of where the center of the scenery LOD radius for this camera will go. This determines where models and builds load and display.

Parameters

lat Latitude of LOD ring center in degrees
lon Longitude of LOD ring center in degrees
alt Altitude of LOD ring center in meters

◆ SetSensorMode()

virtual void SetSensorMode (unsigned int mode) [private] [pure virtual]

Sets the sensor mode of a camera

Parameters

mode Sensor mode (0: None, 1: IR White Hot, 2: IR Black Hot, 3: GData)

◆ SetSideAngles()

virtual void SetSideAngles (float fLeft,
 float fRight,
 float fTop,
 float fBottom
) [private] [pure virtual]

Set Side Angles for a potentially off-axis projection. Positive direction is Right and Down.

Parameters

fLeft Angle in radians from view direction to the left edge of camera's view
fRight Angle in radians from view direction to the right edge of camera's view
fTop Angle in radians from view direction to the top edge of camera's view
fBottom Angle in radians from view direction to bottom left edge of camera's view

◆ SetSinglePassGroupID()

virtual void SetSinglePassGroupID (unsigned int id,
 unsigned int drawOrder
) [private] [pure virtual]

Sets if the camera view should be inside a single pass group

Parameters

[in] **id** for this group, set to 0 to remove single pass grouping
[in] **drawOrder** for this camera view, should be set left to right

Note

Current limit of 2 camera views. DrawOrder example: Left eye view would be 0, right eye view would be 1

Needs to be configured on any type of camera add, remove or view change for the window

◆ SetSmoothRelative6DOF()

virtual void SetSmoothRelative6DOF (float fDeltaX,
 float fDeltaY,
 float fDeltaZ,
 float fPitchDeg,
 float fBankDeg,
 float fHeadingDeg,
 float smoothPanTime = 0
) [private] [pure virtual]

Change the relative position and orientation of camera using a smooth transition

Parameters

fDeltaX,fDeltaY,fDeltaZ 3D position relative to origin
fPitchDeg,fBankDeg,fHeadingDeg Relative orientation pitch bank and heading
smoothPanTime Time-frame for the camera to pan to the new position

◆ SetTargetContainerId()

virtual void SetTargetContainerId (UINT containerId) [private] [pure virtual]

Set the target container ID for the camera to track.

Parameters

containerId The ID of the container to track

◆ SetTargetFrameRate()

virtual void SetTargetFrameRate (const float **targetFrameRate**) **private** **pure virtual**

Set the target frame rate, a rate of 0 is unlimited

Parameters

targetFrameRate target frame rate in seconds

◆ SetTargetLatLonAltDegrees()

virtual void SetTargetLatLonAltDegrees (double **lat**,
double **lon**,
double **alt**
)

private **pure virtual**

Set the target location for the camera to track.

Parameters

lat Target latitude in degrees

lon Target longitude in degrees

alt Target altitude in feet

◆ SetTerrainLODOriginLLA()

virtual void SetTerrainLODOriginLLA (double **lat**,
double **lon**,
double **alt**
)

private **pure virtual**

Set the LLA of where the center of the terrain LOD radius for this camera will go.

Parameters

lat Latitude of LOD ring center in degrees

lon Longitude of LOD ring center in degrees

alt Altitude of LOD ring center in meters

◆ SetUseGlobalTerrainView()

virtual void SetUseGlobalTerrainView (bool **useGlobalTerrainView**) **private** **pure virtual**

Sets whether the camera needs it's own terrain view or can use global

Parameters

[**in**] **useGlobalTerrainView** whether or not the camera will use global terrain view

◆ SetVirtualCockpitTransparentValue()

virtual void SetVirtualCockpitTransparentValue (unsigned int **level**) **private** **pure virtual**

Sets a cockpit transparency level

Parameters

level value between [0-100] to indicate the transparent level. 100 is Fully transparent.

◆ SetZoom()

virtual void SetZoom (float **zoom**) **private** **pure virtual**

Set the Zoom

Parameters

zoom

◆ SetZoomGoal()

virtual void SetZoomGoal (float **zoomGoal**) **private** **pure virtual**

Set the Zoom goal for the camera

Parameters

zoomGoal Zoom goal to set

◆ **TargetCameraLookAt()**

virtual void TargetCameraLookAt() **private** **pure virtual**

Automatically sets the camera to target ground at center of view

◆ **ZoomIn()**

virtual void ZoomIn(void) **private** **pure virtual**

Zoom in one zoom level increment

◆ **ZoomOut()**

virtual void ZoomOut(void) **private** **pure virtual**

Zoom out one zoom level increment

◆ **P3D::IWindowV440**

Window writer interface for setting window states

Inherits IWindowV430.

Private Member Functions

```
virtual void SetDocking (BOOL isDocked) override
virtual void GetPosition (UINT32 &topLeftX, UINT32 &topLeftY) override
virtual void SetPosition (UINT32 x, UINT32 y) override
virtual void SetSize (UINT32 w, UINT32 h) override
virtual void SetPanelOnly (bool bPanelOnly) override
virtual void GetSize (UINT32 &width, UINT32 &height) override
virtual void SetResolution (UINT32 width, UINT32 height) override
virtual void SendWindowMessage (UINT uMsg, long wParam, long lParam) override
virtual void AddPlugin (IWindowPluginV400 *pPlugin) override
virtual void RemovePlugin (IWindowPluginV400 *pPlugin) override
virtual bool IsDocked () const override
virtual IWindowV400 * GetParentWindow () const override
virtual bool IsPanelWindow (UINT32 *pPanelIdent=nullptr) const override
virtual void GetPosition (UINT32 &topLeftX, UINT32 &topLeftY) const override
virtual void GetSize (UINT32 &width, UINT32 &height) const override
virtual const LPCWSTR GetWindowName (void) const override
virtual bool IsActiveWindow (void) const override
virtual bool IsMainAppWindow () const override
virtual bool GetClientToScreen (long &x, long &y) const override
virtual ICameraSystemV400 * GetCameraSystem () override
virtual ICameraSystemV400 * GetPreviousCameraSystem () override
virtual void SetCameraDefinition (const WCHAR *name) override
virtual bool InitWorldSpaceDraw (const ObjectWorldTransform &location, float width, float height) override
virtual void DeinitWorldSpaceDraw () override
virtual void SetWorldSpacePosition (const ObjectWorldTransform &location) override
virtual void GetWorldSpacePosition (ObjectWorldTransform &location) override
virtual void SetWorldSpaceSize (UINT32 width, UINT32 height) override
virtual void SetWorldSpaceSize (float width, float height) override
virtual void SetWorldSpaceFacingFlags (bool pitch, bool bank, bool heading) override
virtual void SetWorldSpaceDepthMode (bool readDepth, bool writeDepth) override
virtual void GetWorldspaceOffset (ObjectLocalTransform &location) override
virtual void SetWorldspaceOffset (ObjectLocalTransform location) override
virtual void GetWorldSpaceSize (float &w, float &h) override
```

```
virtual void GetWorldspaceOffsetReference (CAMERA_TRANSFORM_REFERENCE &reference) override  
virtual void SetWorldspaceOffsetReference (CAMERA_TRANSFORM_REFERENCE reference) override
```

Member Function Documentation

◆ AddPlugin()

```
virtual void AddPlugin ( IWindowPluginV400 * pPlugin ) [private] [pure virtual]
```

Add a plugin to the window.

Parameters

pPlugin the plugin to add

◆ DeinitWorldSpaceDraw()

```
virtual void DeinitWorldSpaceDraw ( ) [private] [pure virtual]
```

Shutdown the world space draw object, this will cause the window to no longer draw in world space.

◆ GetCameraSystem()

```
virtual ICameraSystemV400* GetCameraSystem ( ) [private] [pure virtual]
```

Get camera system attached to this window.

Returns

Camera System attached to this window or nullptr if none exists.

◆ GetClientToScreen()

```
virtual bool GetClientToScreen ( long & x,  
                                long & y  
                                ) [const] [private] [pure virtual]
```

Get the screen coords for the given client coords.

Parameters

[in/out] x Client x coordinate
[in/out] y Client y coordinate

Returns

If the function executed successfully

◆ GetParentWindow()

```
virtual IWindowV400* GetParentWindow ( ) const [private] [pure virtual]
```

Get the parent window

Returns

IWindowV400 parent to this window or nullptr if none exists.

◆ GetPosition() [1/2]

```
virtual void GetPosition ( UINT32 & topLeftX,  
                          UINT32 & topLeftY  
                          ) [private] [pure virtual]
```

Get the position of a window in pixels

Parameters

width,height Window size in pixels

◆ GetPosition() [2/2]

```
virtual void GetPosition ( UINT32 & topLeftX,  
                          UINT32 & topLeftY
```

) const private pure virtual

Get the position of the window in screen space

Parameters

[out] **topLeftX,topLeftY** position of the window in screen space

◆ **GetPreviousCameraSystem()**

virtual ICameraSystemV400* GetPreviousCameraSystem() const private pure virtual

Get previous camera system attached to this window. This can be useful when handling a view changes triggered by the user or by calls to SetCameraDefinition.

Returns

Camera System previously attached to this window or nullptr if none exists.

◆ **GetSize() [1/2]**

virtual void GetSize(**UINT32 & width,**
UINT32 & height
) const private pure virtual

Get the size of a window in pixels

Parameters

width,height Window size in pixels

◆ **GetSize() [2/2]**

virtual void GetSize(**UINT32 & width,**
UINT32 & height
) const private pure virtual

Get the size of the window in screen space

Parameters

[out] **width,height** size of the window in screen space

◆ **GetWindowName()**

virtual const LPCWSTR GetWindowName(void) const private pure virtual

Get the window name

Returns

window name

◆ **GetWorldspaceOffset()**

virtual void GetWorldspaceOffset(**ObjectLocalTransform & location**) const private pure virtual

Get the local transform of the object used to calculate the world space offset

◆ **GetWorldspaceOffsetReference()**

virtual void GetWorldspaceOffsetReference(**CAMERA_TRANFORM_REFERENCE & reference**) const private pure virtual

Get the reference the world space draw object uses to offset its position and PBH

◆ **GetWorldSpacePosition()**

virtual void GetWorldSpacePosition(**ObjectWorldTransform & location**) const private pure virtual

Get the world space location of the draw object if one exist for this window.

Parameters

location World location to store the world space location.

◆ **GetWorldSpaceSize()**

```
virtual void GetWorldSpaceSize ( float & x,  
                               float & y  
                           )
```

private **pure virtual**

Get the size of the world space draw object.

Parameters

[in/out] **x** Draw Objects Size X
[in/out] **y** Draw Objects Size Y

◆ **InitWorldSpaceDraw()**

```
virtual bool InitWorldSpaceDraw ( const ObjectWorldTransform & location,  
                                float width,  
                                float height  
                           )
```

private **pure virtual**

Setup a world space draw object for the window, the object is a double sided plane. This is useful for displaying windows in world space locations.

Parameters

location World location to place the draw object.
width,height World plane size in meters.

Returns

If the creation of the world space draw object was successful.

◆ **IsActiveWindow()**

```
virtual bool IsActiveWindow ( void ) const
```

private **pure virtual**

Check if window's active camera is in focus

Returns

true if window's active camera is in focus

◆ **IsDocked()**

```
virtual bool IsDocked ( ) const
```

private **pure virtual**

Get window's docking state

Returns

bool value of true if window is docked and false if not

◆ **IsMainAppWindow()**

```
virtual bool IsMainAppWindow ( ) const
```

private **pure virtual**

Is this window the main Application Window?

Returns

true if this window is the main window

◆ **IsPanelWindow()**

```
virtual bool IsPanelWindow ( UINT32 * pPanelId = nullptr ) const
```

private **pure virtual**

Get window's panel state

Parameters

[out] **pPanelId** optional panel id if this window is a panel

Returns

bool value of true if window is a panel and false if not

◆ **RemovePlugin()**

```
virtual void RemovePlugin ( IWindowPluginV400 * pPlugin )
```

private **pure virtual**

Remove a plugin from the window.

Parameters

pPlugin the plugin to remove

◆ **SendWindowMessage()**

```
virtual void SendWindowMessage ( UINT uMsg,  
                                long wParam,  
                                long lParam  
                            )
```

private **pure virtual**

Send a message to the window.

Parameters

uMsg The message for the window
wParam w param of the message
lParam l param of the message

◆ **SetCameraDefinition()**

```
virtual void SetCameraDefinition ( const WCHAR * name ) private pure virtual
```

Set the window to use the specified camera. This will get or create an instance of a camera using the specified camera definition name.

Parameters

name Name of camera definition

Remarks

camera definitions are defined in Cameras.cfg or in aircraft.cfg. This will cause the camera to change, so calling **GetCameraSystem()** window will return a different value. The current camera instance will still be maintained until the window that created it has been closed, and it can be accessed by calling GetPreviousCameraSystem after the changed.

◆ **SetDocking()**

```
virtual void SetDocking ( BOOL isDocked ) private pure virtual
```

Set Docked/Undocked state

Parameters

isDocked Set TRUE to dock or FALSE to undock a window

◆ **SetPanelOnly()**

```
virtual void SetPanelOnly ( bool bPanelOnly ) private pure virtual
```

Set this window to panel only mode

Parameters

bPanelOnly panel-only mode on or off

◆ **SetPosition()**

```
virtual void SetPosition ( UINT32 x,  
                           UINT32 y  
                       )
```

private **pure virtual**

Set the position of the window in screen space

Parameters

x,y Screenspace position of window

◆ **SetResolution()**

```
virtual void SetResolution ( UINT32 width,  
                           UINT32 height  
                       )
```

private **pure virtual**

Set the size render target resolution in pixels. This can be used to super-sample or subsample on-screen views by setting the resolution different than the window size.

Parameters

width,height Window size in pixels

◆ SetSize()

```
virtual void SetSize( UINT32 w,  
                      UINT32 h  
                  )
```

private **pure virtual**

Set the size of a window in pixels

Parameters

width,height Window size in pixels

◆ SetWorldSpaceDepthMode()

```
virtual void SetWorldSpaceDepthMode( bool readDepth,  
                                    bool writeDepth  
                                )
```

private **pure virtual**

Set the world space depth mode of the draw object if one exist for this window.

Parameters

readDepth,writeDepth Flag for setting depth read write status.

◆ SetWorldSpaceFacingFlags()

```
virtual void SetWorldSpaceFacingFlags( bool pitch,  
                                       bool bank,  
                                       bool heading  
                                     )
```

private **pure virtual**

Set the world space size of the draw object if one exist for this window.

Parameters

pitch,bank,heading Flag for setting facing status.

◆ SetWorldspaceOffset()

```
virtual void SetWorldspaceOffset( ObjectLocalTransform location )
```

private **pure virtual**

Set the local transfrom of the object used to calculate the world space offset

◆ SetWorldspaceOffsetReference()

```
virtual void SetWorldspaceOffsetReference( CAMERA_TRANSFORM_REFERENCE reference )
```

private **pure virtual**

Set the reference the world space draw object uses to offset its position and PBH

◆ SetWorldSpacePosition()

```
virtual void SetWorldSpacePosition( const ObjectWorldTransform & location )
```

private **pure virtual**

Set the world space location of the draw object if one exist for this window.

Parameters

location World location to place the draw object.

◆ SetWorldSpaceSize() [1 / 2]

```
virtual void SetWorldSpaceSize( UINT32 width,  
                               UINT32 height  
                             )
```

private **pure virtual**

Set the world space size of the draw object if one exist for this window.

Parameters

width,height World plane size in meters.

◆ SetWorldSpaceSize() [2/2]

```
virtual void SetWorldSpaceSize ( float width,  
                               float height  
                           )  
    private pure virtual
```

Set the world space size of the draw object if one exist for this window.

Parameters

width,height World plane size in meters.

◆ P3D::WindowPlugin

class P3D::WindowPlugin

Base implementation of the IWindowPluginV400 which defines default implementations of all IWindowPluginV400 functions and a default IUnknown implementation.

Inherits IWindowPluginV400.

Public Member Functions

```
    WindowPlugin () noexcept  
    virtual ~WindowPlugin ()  
    virtual void OnAdd (IWindowV400 *pWindow, ICameraSystemV400 *pCamera) override  
    virtual void OnRemove (IWindowV400 *pWindow, ICameraSystemV400 *pCamera) override  
    virtual void OnPreCameraUpdate (IWindowV400 *pWindow, ICameraSystemV400  
                                   *pCamera) override  
    virtual void OnPostCameraUpdate (IWindowV400 *pWindow, ICameraSystemV400  
                                   *pCamera) override  
    virtual void OnViewChange (IWindowV400 *pWindow, ICameraSystemV400  
                             *pCamera) override  
    virtual void OnClose (IWindowV400 *pWindow, ICameraSystemV400 *pCamera)  
        override  
    virtual bool OnUserInput (HWND wnd, UINT message, WPARAM wParam, LPARAM  
                           lParam) override  
    DEFAULT_REFCOUNT_INLINE_IMPL ()  
STDMETHODIMP QueryInterface (REFIID iid, PVOID *ppv)
```

Constructor & Destructor Documentation

◆ WindowPlugin()

```
WindowPlugin ( ) inline noexcept
```

◆ ~WindowPlugin()

```
virtual ~WindowPlugin ( ) inline virtual
```

Member Function Documentation

◆ DEFAULT_REFCOUNT_INLINE_IMPL()

```
DEFAULT_REFCOUNT_INLINE_IMPL ( )
```

◆ OnAdd()

```
virtual void OnAdd (IWindowV400 *  
                    pWindow,  
                    ICameraSystemV400 *  
                    pCamera  
                  )  
    inline override virtual
```

◆ OnClose()

```
virtual void OnClose (IWindowV400 *  
                     pWindow,  
                     ICameraSystemV400 *  
                     pCamera  
                   )  
    inline override virtual
```

◆ OnPostCameraUpdate()

```
virtual void OnPostCameraUpdate ( IWindowV400 * pWindow,  
                                ICameraSystemV400 * pCamera  
                            )
```

inline override virtual

◆ OnPreCameraUpdate()

```
virtual void OnPreCameraUpdate ( IWindowV400 * pWindow,  
                                ICameraSystemV400 * pCamera  
                            )
```

inline override virtual

◆ OnRemove()

```
virtual void OnRemove ( IWindowV400 * pWindow,  
                       ICameraSystemV400 * pCamera  
                     )
```

inline override virtual

◆ OnUserInput()

```
virtual bool OnUserInput ( HWND wnd,  
                          UINT message,  
                          WPARAM wParam,  
                          LPARAM lParam  
                        )
```

inline override virtual

◆ OnViewChange()

```
virtual void OnViewChange ( IWindowV400 * pWindow,  
                           ICameraSystemV400 * pCamera  
                         )
```

inline override virtual

◆ QueryInterface()

```
STDMETHODIMP QueryInterface ( REFIID riid,  
                             PVOID * ppv  
                           )
```

inline

Typedefs

```
using IWindowList = IListBuilder< IWindowV400 >  
using WindowList = CComPtrVecBuilder< IWindowV400 >  
typedef IWindowPluginSystemV440 IWindowPluginSystem  
typedef IWindowPluginV400 IWindowPlugin  
typedef IWindowV440 IWindow  
typedef ICameraSystemV451 ICameraSystem
```

Enumerations

```
enum MOUSE_CLICK_TYPE {  
    MOUSE_CLICK_NONE, MOUSE_CLICK_RIGHT_SINGLE,  
    MOUSE_CLICK_MIDDLE_SINGLE, MOUSE_CLICK_LEFT_SINGLE,  
    MOUSE_CLICK_RIGHT_DOUBLE, MOUSE_CLICK_MIDDLE_DOUBLE,  
    MOUSE_CLICK_LEFT_DOUBLE, MOUSE_CLICK_RIGHT_DRAG,  
    MOUSE_CLICK_MIDDLE_DRAG, MOUSE_CLICK_LEFT_DRAG,  
    MOUSE_CLICK_MOVE, MOUSE_CLICK_RIGHT_RELEASE,  
    MOUSE_CLICK_MIDDLE_RELEASE, MOUSE_CLICK_LEFT_RELEASE,  
    MOUSE_CLICK_WHEEL_UP, MOUSE_CLICK_WHEEL_DOWN,  
    MOUSE_CLICK_LEAVE, MOUSE_CLICK_COUNT  
}  
enum HMD_MODE { HMD_MODE_NONE, HMD_MODE_VR, HMD_MODE_AR }  
enum CAMERA_TRANSFORM_REFERENCE { CAMERA_REFERENCE_ORIGIN = 0,  
                                 CAMERA_REFERENCE_OFFSET, CAMERA_REFERENCE_EYE,  
                                 CAMERA_REFERENCE_NONE }  
enum CAMERA_SYSTEM_ORIGIN {  
    CAMERA_SYSTEM_ORIGIN_UNKNOWN = 0,  
    CAMERA_SYSTEM_ORIGIN_OBJECT_COCKPIT,  
    CAMERA_SYSTEM_ORIGIN_OBJECT_VIRTUAL_COCKPIT,  
    CAMERA_SYSTEM_ORIGIN_OBJECT_CENTER,  
    CAMERA_SYSTEM_ORIGIN_OBJECT_CENTER_NO_ORIENT,  
    CAMERA_SYSTEM_ORIGIN_OBJECT_PILOT,  
    CAMERA_SYSTEM_ORIGIN_TOWER, CAMERA_SYSTEM_ORIGIN_FIXED,  
    CAMERA_SYSTEM_ORIGIN_WORLD_OBJECT,
```

```

CAMERA_SYSTEM_ORIGIN_SCENARIO,
CAMERA_SYSTEM_ORIGIN_LATLONALT_ORTHOGONAL,
CAMERA_SYSTEM_ORIGIN_OBSERVER,
CAMERA_SYSTEM_ORIGIN_OBJECT_AI_VIRTUAL_COCKPIT,
CAMERA_SYSTEM_ORIGIN_OBJECT_AI_CENTER,
CAMERA_SYSTEM_ORIGIN_ATTACH_POINT,
CAMERA_SYSTEM_ORIGIN_WORLD_OBJECT_VIRTUAL_COCKPIT,
CAMERA_SYSTEM_ORIGIN_COUNT
}

enum CAMERA_SYSTEM_CATEGORY {
    CAMERA_SYSTEM_CATEGORY_UNKNOWN = 0,
    CAMERA_SYSTEM_CATEGORY_COCKPIT,
    CAMERA_SYSTEM_CATEGORY_OUTSIDE,
    CAMERA_SYSTEM_CATEGORY_TOWER,
    CAMERA_SYSTEM_CATEGORY_RUNWAY,
    CAMERA_SYSTEM_CATEGORY_AIRCRAFT,
    CAMERA_SYSTEM_CATEGORY_AIR_TRAFFIC,
    CAMERA_SYSTEM_CATEGORY_MULTI_PLAYER,
    CAMERA_SYSTEM_CATEGORY_SCENERY,
    CAMERA_SYSTEM_CATEGORY_IG,
    CAMERA_SYSTEM_CATEGORY_OBSERVER,
    CAMERA_SYSTEM_CATEGORY_SENSOR,
    CAMERA_SYSTEM_CATEGORY_WEAPON,
    CAMERA_SYSTEM_CATEGORY_VR, CAMERA_SYSTEM_CATEGORY_CUSTOM,
    CAMERA_SYSTEM_CATEGORY_DIS_ENTITIES,
    CAMERA_SYSTEM_CATEGORY_HLA_OBJECTS,
    CAMERA_SYSTEM_CATEGORY_EQUIPMENT,
    CAMERA_SYSTEM_CATEGORY_FIRST_PERSON,
    CAMERA_SYSTEM_CATEGORY_THIRD_PERSON,
    CAMERA_SYSTEM_CATEGORY_SCENARIO_OBJECTS,
    CAMERA_SYSTEM_CATEGORY_COUNT
}

```

Functions

```

virtual void OnRemove (IWindowV400 *pWindow, ICameraSystemV400 *pCamera) override
virtual void OnPreCameraUpdate (IWindowV400 *pWindow, ICameraSystemV400 *pCamera)
override
virtual void OnPostCameraUpdate (IWindowV400 *pWindow, ICameraSystemV400 *pCamera)
override
virtual void OnViewChange (IWindowV400 *pWindow, ICameraSystemV400 *pCamera) override
virtual void OnClose (IWindowV400 *pWindow, ICameraSystemV400 *pCamera) override
virtual bool OnUserInput (HWND wnd, UINT message, WPARAM wParam, LPARAM lParam)
override

```

Variables

```

interface ICameraSystemV400 *pCamera abstract
    REFIID IID_IWindowPluginSystemV440 =
        __uuidof(IWindowPluginSystemV440)
    REFIID IID_IWindowPluginV400 =
        __uuidof(IWindowPluginV400)
    REFIID IID_IWindowV440 = __uuidof(IWindowV440)
    REFIID IID_ICameraSystemV451 =
        __uuidof(ICameraSystemV451)
    REFIID SID_WindowPluginSystem =
        __uuidof(WindowPluginSystemV400)
    REFIID IID_IWindowPluginSystem =
        IID_IWindowPluginSystemV440
    REFIID IID_IWindowPlugin = IID_IWindowPluginV400
    REFIID IID_ICameraSystem = IID_ICameraSystemV451
    REFIID IID_IWindow = IID_IWindowV440

```

Typedef Documentation

◆ ICameraSystem

```
typedef ICameraSystemV451 ICameraSystem
```

◆ IWindow

```
typedef IWindowV440 IWindow
```

◆ IWindowList

```
using IWindowList = IListBuilder<IWindowV400>
```

◆ IWindowPlugin

```
typedef IWindowPluginV400 IWindowPlugin
```

◆ IWindowPluginSystem

```
typedef IWindowPluginSystemV440 IWindowPluginSystem
```

◆ WindowList

```
using WindowList = CComPtrVecBuilder<IWindowV400>
```

[Enumeration Type Documentation](#)

◆ CAMERA_SYSTEM_CATEGORY

```
enum CAMERA_SYSTEM_CATEGORY
```

Enumerator

CAMERA_SYSTEM_CATEGORY_UNKNOWN
CAMERA_SYSTEM_CATEGORY_COCKPIT
CAMERA_SYSTEM_CATEGORY_OUTSIDE
CAMERA_SYSTEM_CATEGORY_TOWER
CAMERA_SYSTEM_CATEGORY_RUNWAY
CAMERA_SYSTEM_CATEGORY_AIRCRAFT
CAMERA_SYSTEM_CATEGORY_AIR_TRAFFIC
CAMERA_SYSTEM_CATEGORY_MULTI_PLAYER
CAMERA_SYSTEM_CATEGORY_SCENERY
CAMERA_SYSTEM_CATEGORY_IG
CAMERA_SYSTEM_CATEGORY_OBSERVER
CAMERA_SYSTEM_CATEGORY_SENSOR
CAMERA_SYSTEM_CATEGORY_WEAPON
CAMERA_SYSTEM_CATEGORY_VR
CAMERA_SYSTEM_CATEGORY_CUSTOM
CAMERA_SYSTEM_CATEGORY_DIS_ENTITIES
CAMERA_SYSTEM_CATEGORY_HLA_OBJECTS
CAMERA_SYSTEM_CATEGORY_EQUIPMENT
CAMERA_SYSTEM_CATEGORY_FIRST_PERSON
CAMERA_SYSTEM_CATEGORY_THIRD_PERSON
CAMERA_SYSTEM_CATEGORY_SCENARIO_OBJECTS
CAMERA_SYSTEM_CATEGORY_COUNT

◆ CAMERA_SYSTEM_ORIGIN

```
enum CAMERA_SYSTEM_ORIGIN
```

Enumerator

CAMERA_SYSTEM_ORIGIN_UNKNOWN
CAMERA_SYSTEM_ORIGIN_OBJECT_COCKPIT
CAMERA_SYSTEM_ORIGIN_OBJECT_VIRTUAL_COCKPIT
CAMERA_SYSTEM_ORIGIN_OBJECT_CENTER
CAMERA_SYSTEM_ORIGIN_OBJECT_CENTER_NO_ORIENT
CAMERA_SYSTEM_ORIGIN_OBJECT_PILOT
CAMERA_SYSTEM_ORIGIN_TOWER
CAMERA_SYSTEM_ORIGIN_FIXED
CAMERA_SYSTEM_ORIGIN_WORLD_OBJECT
CAMERA_SYSTEM_ORIGIN_SCENARIO
CAMERA_SYSTEM_ORIGIN_LATLONALT_ORTHOGONAL
CAMERA_SYSTEM_ORIGIN_OBSERVER
CAMERA_SYSTEM_ORIGIN_OBJECT_AI_VIRTUAL_COCKPIT
CAMERA_SYSTEM_ORIGIN_OBJECT_AI_CENTER
CAMERA_SYSTEM_ORIGIN_ATTACH_POINT
CAMERA_SYSTEM_ORIGIN_WORLD_OBJECT_VIRTUAL_COCKPIT
CAMERA_SYSTEM_ORIGIN_COUNT

◆ CAMERA_TRANSFORM_REFERENCE

enum CAMERA_TRANSFORM_REFERENCE

Enumerator

CAMERA_REFERENCE_ORIGIN
CAMERA_REFERENCE_OFFSET
CAMERA_REFERENCE_EYE
CAMERA_REFERENCE_NONE

◆ HMD_MODE

enum HMD_MODE

Enumerator

HMD_MODE_NONE
HMD_MODE_VR
HMD_MODE_AR

◆ MOUSE_CLICK_TYPE

enum MOUSE_CLICK_TYPE

Enumerator

MOUSE_CLICK_NONE
MOUSE_CLICK_RIGHT_SINGLE
MOUSE_CLICK_MIDDLE_SINGLE
MOUSE_CLICK_LEFT_SINGLE
MOUSE_CLICK_RIGHT_DOUBLE
MOUSE_CLICK_MIDDLE_DOUBLE
MOUSE_CLICK_LEFT_DOUBLE
MOUSE_CLICK_RIGHT_DRAG
MOUSE_CLICK_MIDDLE_DRAG
MOUSE_CLICK_LEFT_DRAG
MOUSE_CLICK_MOVE
MOUSE_CLICK_RIGHT_RELEASE
MOUSE_CLICK_MIDDLE_RELEASE
MOUSE_CLICK_LEFT_RELEASE
MOUSE_CLICK_WHEEL_UP
MOUSE_CLICK_WHEEL_DOWN
MOUSE_CLICK_LEAVE
MOUSE_CLICK_COUNT

Function Documentation

◆ OnClose()

```
virtual void P3D::OnClose ( IWindowV400 * pWindow,  
                           ICameraSystemV400 * pCamera  
                           )
```

pure virtual

Called when the window closes.

Parameters

pWindow Window the plugin is currently operating on
pCamera Camera the plugin is currently operating on

◆ OnPostCameraUpdate()

```
virtual void P3D::OnPostCameraUpdate ( IWindowV400 * pWindow,  
                                      ICameraSystemV400 * pCamera  
                                      )
```

pure virtual

Called after the camera updates. This is the best place to set values that override a camera update. SetCameraLODOrginLLA for example must be called here because the lod origin is reset to the camera's lla each frame in the camer's update.

Parameters

pWindow Window the plugin is currently operating on
pCamera Camera the plugin is currently operating on

◆ OnPreCameraUpdate()

```
virtual void P3D::OnPreCameraUpdate ( IWindowV400 * pWindow,  
                                     ICameraSystemV400 * pCamera  
                                     )
```

pure virtual

Called before the camera updates. This is the best place to set most camera values because many of them are used to update the camera's position and orientation in the world

Parameters

pWindow Window the plugin is currently operating on
pCamera Camera the plugin is currently operating on

◆ OnRemove()

```
virtual void P3D::OnRemove ( IWindowV400 * pWindow,  
                           ICameraSystemV400 * pCamera  
                           )
```

pure virtual

Called when the plugin is removed from a window.

Parameters

pWindow Window the plugin is currently operating on
pCamera Camera the plugin is currently operating on

◆ OnUserInput()

```
virtual bool P3D::OnUserInput ( HWND wnd,  
                             UINT message,  
                             WPARAM wParam,  
                             LPARAM lParam  
                             )
```

pure virtual

Called when mouse or keyboard input is passed to the window.

Parameters

wnd HWND for the windows message
message Message ID for the windows message
wParam WPARAM for the windows message
lParam LPARAM for the windows message

Returns

true if the message was handled and should not be passed through to any windows underneath.

◆ OnViewChange()

```
virtual void P3D::OnViewChange ( IWindowV400 * pWindow,  
                               ICameraSystemV400 * pCamera  
                               )
```

pure virtual

Called after window changes cameras.

Parameters

pWindow Window the plugin is currently operating on
pCamera Camera the plugin is currently operating on

Variable Documentation

◆ abstract

interface ICameraSystemV400* pCamera abstract

◆ IID_ICameraSystem

REFIID IID_ICameraSystem = IID_ICameraSystemV451

◆ IID_ICameraSystemV451

REFIID IID_ICameraSystemV451 = __uuidof(ICameraSystemV451)

◆ IID_IWindow

REFIID IID_IWindow = **IID_IWindowV440**

◆ IID_IWindowPlugin

REFIID IID_IWindowPlugin = **IID_IWindowPluginV400**

◆ IID_IWindowPluginSystem

REFIID IID_IWindowPluginSystem = **IID_IWindowPluginSystemV440**

◆ IID_IWindowPluginSystemV440

REFIID IID_IWindowPluginSystemV440 = __uuidof(**IWindowPluginSystemV440**)

◆ IID_IWindowPluginV400

REFIID IID_IWindowPluginV400 = __uuidof(**IWindowPluginV400**)

◆ IID_IWindowV440

REFIID IID_IWindowV440 = __uuidof(**IWindowV440**)

◆ SID_WindowPluginSystem

REFIID SID_WindowPluginSystem = __uuidof(**IWindowPluginSystemV400**)

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Event Service

Overview

This service is provided through the PDK interface and allows callers to register for events which are signaled by Prepar3D. Event GUIDs are defined in the interface header. Currently supported events are:

- EVENTID_Frame: This event is signaled once per frame.
- EVENTID_1Hz: This event is signaled once per second.
- EVENTID_CustomObjectDraw: This event is signaled once per view each frame.

See [PdkPlugin](#) for an example of registering for multiple events and messages.

Classes

- struct [P3DParameter](#)
class [IParameterListV400](#)
class [ICallbackV400](#)
class [IEventServiceV400](#)
-

Class Documentation

◆ [P3D::P3DParameter](#)

struct P3D::P3DParameter

Parameter values used for [IParameterListV400](#)

Class Members

UINT64	Value
------------------------	-------

◆ [P3D::IParameterListV400](#)

```
class P3D::IParameterListV400
```

Parameter list interface used by [ICallbackV400](#)

Inherits [IUnknown](#).

Inherited by [ParameterList](#).

Private Member Functions

```
virtual IServiceProvider* GetServiceProvider() override  
virtual P3DParameter GetParameter(UINT32 index) override  
virtual UINT32 GetCount() override
```

Member Function Documentation

◆ **GetCount()**

```
virtual UINT32 GetCount( ) private pure virtual
```

Get parameter count.

Parameters

index parameter index;

Returns

number of parameters in the params list.

Implemented in [ParameterList](#).

◆ **GetParameter()**

```
virtual P3DParameter GetParameter( UINT32 index ) private pure virtual
```

Get [P3DParameter](#) by index.

Parameters

index parameter index;

Returns

The parameter requested.

Implemented in [ParameterList](#).

◆ **GetServiceProvider()**

```
virtual IServiceProvider* GetServiceProvider( ) private pure virtual
```

Get service provider. Use QueryService to request services from it.

Returns

Currently all callbacks pass an [IPdk](#) pointer as the service provider, but some callbacks may provide callback-specific providers in the future.

Implemented in [ParameterList](#).

◆ P3D::ICallbackV400

class P3D::ICallbackV400

Callback interface which must be implemented to register for events.

Parameters

index parameter index;

Returns

number of parameters in the params list.

Inherits [IUnknown](#).

Inherited by [P3dCallback](#), and [PdkPlugin::P3dPluginCallback](#).

Private Member Functions

virtual void [Invoke](#) ([IParameterListV400](#) *pParams) override

Member Function Documentation

◆ [Invoke\(\)](#)

virtual void [Invoke](#) ([IParameterListV400](#) * pParams) private pure virtual

Called when the registered event is fired.

Parameters

pParams Callback parameters interface pointer.

Implemented in [PdkPlugin::MessageCallback](#), [PdkPlugin::CustomRenderCallback](#), [PdkPlugin::FrameCallback](#), [PdkPlugin::OneHzCallback](#), [PdkPlugin::P3dPluginCallback](#), and [P3dCallback](#).

◆ P3D::IEventServiceV400

class P3D::IEventServiceV400

Service used to register for Prepar3D event callbacks

Inherits [IUnknown](#).

Private Member Functions

```
virtual HRESULT RegisterCallback (const GUID &eventID, ICallbackV400 *pCallback)
override
virtual HRESULT UnregisterCallback (const GUID &eventID, ICallbackV400 *pCallback)
override
virtual HRESULT SendMessageEvent (UINT32 messageID, PVOID messageParam) override
```

Member Function Documentation

◆ RegisterCallback()

```
virtual HRESULT RegisterCallback ( const GUID & eventID,
ICallbackV400 * pCallback
)
```

private pure virtual

Register an event callback.

Parameters

eventID GUID id of the event to register for
pCallback Callback to register

Returns

S_OK if succeeded and E_FAIL if it failed.

◆ SendMessageEvent()

```
virtual HRESULT SendMessageEvent ( UINT32 messageID,
PVOID messageParam
)
```

private pure virtual

Send a message event.

Parameters

messageID message ID
PVOID message parameter

◆ UnregisterCallback()

```
virtual HRESULT UnregisterCallback ( const GUID & eventID,
ICallbackV400 * pCallback
)
```

private pure virtual

Unregister an event callback.

Parameters

eventID GUID id of the event to register for

pCallback Callback to register

Returns

S_OK if succeeded and E_FAIL if it failed.

Macros

#define **EVENT_MESSAGE_APPLICATION_STARTUP** 0x0036
Application is starting up. [More...](#)

#define **EVENT_MESSAGE_APPLICATION_SHUTDOWN** 0x0037
Application is shutting down. [More...](#)

#define **EVENT_MESSAGE_APPLICATION_RESTART** 0x003a
Application is being "restarted" by a second instance. [More...](#)

#define **EVENT_MESSAGE_APPLICATION_ACTIVATED** 0x003b
Application just received WM_ACTIVATEAPP. [More...](#)

#define **EVENT_MESSAGE_APPLICATION_DEACTIVATED** 0x003c
Application just received WM_ACTIVATEAPP. [More...](#)

#define **EVENT_MESSAGE_MAIN_WINDOW_CHANGE** 0x003d
Application main window maximized, etc. [More...](#)

#define **EVENT_MESSAGE_MENU_MODE_START** 0x0040
Main menu is getting ready to be displayed. [More...](#)

#define **EVENT_MESSAGE_MENU_MODE_STOP** 0x0041
Main menu is done being displayed. [More...](#)

#define **EVENT_MESSAGE_CONTEXT_MENU_MODE_START** 0x0042
Context menu is getting ready to be displayed. [More...](#)

#define **EVENT_MESSAGE_CONTEXT_MENU_MODE_STOP** 0x0043
Context menu is done being displayed. [More...](#)

#define **EVENT_MESSAGE_DIALOG_MODE_START** 0x0044
A dialogue is getting ready to be displayed. [More...](#)

#define **EVENT_MESSAGE_DIALOG_MODE_STOP** 0x0045
A dialogue is done being displayed. [More...](#)

#define **EVENT_MESSAGE_MEMORY_LOW** 0x0046
A malloc has failed, if you see this free up some memory. [More...](#)

#define **EVENT_MESSAGE_DEVICE_RESET_COMPLETED** 0x0181
Rendering system finished remapping windows to devices. [More...](#)

#define **EVENT_MESSAGE_MAIN_LOADING_STATE_CHANGE** 0x0191
On MainLoadingState change send new state. [More...](#)

#define **EVENT_MESSAGE_LOADING_COMPLETE** 0x0197
Done loading. [More...](#)

#define **EVENT_MESSAGE_APPLICATION_DLLS_LOADED** 0x003e
Application dlls and plugin dlls are done loading. [More...](#)

#define **EVENT_MESSAGE_SIM_PAUSED** 0x0038
Simulation is paused. [More...](#)

#define **EVENT_MESSAGE_SIM_UNPAUSED** 0x0039

Simulation is unpause. [More...](#)

#define **EVENT_MESSAGE_SIMCONNECT_SIMULATION_START** 0x0022

SimConnect has detected "simulating" has started. [More...](#)

#define **EVENT_MESSAGE_SIMCONNECT_SIMULATION_STOP** 0x0023

SimConnect has detected "simulating" has stopped. [More...](#)

#define **EVENT_MESSAGE_SCENARIO_LOADING** 0x0027

Sent at beginning of scenario load sequence. [More...](#)

#define **EVENT_MESSAGE_SCENARIO_LOADED** 0x0028

Sent at end of scenario load sequence. [More...](#)

#define **EVENT_MESSAGE_CURRENT_SCENARIO_CHANGED** 0x002F

Global current scenario changed. [More...](#)

#define **EVENT_MESSAGE_SCENARIO_POST_LOADED** 0x0030

After scenario PostLoad is completed. [More...](#)

#define **EVENT_MESSAGE_CRASH_START** 0x0081

The user's plane just started crash sequence. [More...](#)

#define **EVENT_MESSAGE_CRASH_COMPLETE** 0x0082

The user plane crashed and the crash sequence is done. [More...](#)

#define **EVENT_MESSAGE_TIMESEASON_CHANGED_10MIN** 0x0110

10 minute time change [More...](#)

#define **EVENT_MESSAGE_TIMESEASON_CHANGED_1MIN** 0x0112

1 minute time change [More...](#)

#define **EVENT_MESSAGE_TIMESEASON_CHANGED_BY_USER** 0x0113

User has changed the time via the time/season dialogue. [More...](#)

#define **EVENT_MESSAGE_REQUESTED_SIM_RATE_CHANGED** 0x011A

Sends the new value of the sim speed. [More...](#)

#define **EVENT_MESSAGE_WEATHER_CHANGED** 0x0133

If there's any change in the weather, this message will get sent. [More...](#)

#define **EVENT_MESSAGE_WEATHER_MODE_CHANGED** 0x0138

Weather system mode (i.e. RWW, theme, custom) has been changed. [More...](#)

#define **EVENT_MESSAGE_CHANGE_USER_VEHICLE_START** 0x240

User vehicle is changing. [More...](#)

#define **EVENT_MESSAGE_CHANGE_USER_VEHICLE_CREATED** 0x241

New User Vehicle is created but hasn't been registered. [More...](#)

#define **EVENT_MESSAGE_CHANGE_USER_VEHICLE_FINISH** 0x242

User vehicle has finished changing. [More...](#)

#define **EVENT_MESSAGE_SIM_FROZEN** 0x270

Sim frozen. [More...](#)

#define **EVENT_MESSAGE_SIM_UNFROZEN** 0x271

Sim unfrozen. [More...](#)

#define **EVENT_MESSAGE_AVATAR_ATTACH_CHANGED** 0x280

The user's avatar has either attached or detached. [More...](#)

#define **EVENT_MESSAGE_DESTROY_WINDOW** 0x0150

Window is about to be destroyed. [More...](#)

#define	EVENT_MESSAGE_CREATE_WINDOW	0x0151
New window was created. More...		
#define	EVENT_MESSAGE_VIEW_ACTIVATE	0x0160
Window activated. More...		
#define	EVENT_MESSAGE_VIEW_MODE_CHANGED	0x0161
View changed on a window. More...		
#define	EVENT_MESSAGE_VIEW_CREATE	0x0163
Window created. More...		
#define	EVENT_MESSAGE_VIEW_DESTROY	0x0164
Window destroyed. More...		
#define	EVENT_MESSAGE_MULTIPLAYER_SERVER_STARTED	0x0192
When the host has created the server. More...		
#define	EVENT_MESSAGE_MULTIPLAYER_CLIENT_STARTED	0x0193
When a client has joined a session. More...		
#define	EVENT_MESSAGE_MULTIPLAYER_LAUNCHED	0x0194
When the multiplayer session is launched. More...		
#define	EVENT_MESSAGE_MULTIPLAYER_SESSION_ENDED	0x0195
When the multiplayer session is ended. More...		
#define	EVENT_MESSAGE_HMD_CONNECTED	0x01A3
Sent when an HMD is connected. More...		
#define	EVENT_MESSAGE_HMD_DISCONNECTED	0x01A4
Sent when an HMD is disconnected. More...		
#define	EVENT_MESSAGE_HMD_VIEW_OPENED	0x01A5
Sent when an HMD view is turned on. More...		
#define	EVENT_MESSAGE_HMD_VIEW_CLOSED	0x01A6
Sent when an HMD view is turned off. More...		
#define	EVENT_MESSAGE_HMD_VIEW_DISABLE	0x01A7
Send to disable current HMD view and revert to previous. More...		
#define	EVENT_MESSAGE_PANEL_OPENED	0x0231
Panel opened. More...		
#define	EVENT_MESSAGE_PANEL_CLOSED	0x0232
Panel closed. More...		
#define	EVENT_MESSAGE_SIMDIRECTOR_ACTIVE	0x022D
#define	EVENT_MESSAGE_SIMDIRECTOR_INACTIVE	0x022E
#define	EVENT_MESSAGE_ENABLE_VR_ON_STARTUP	0x300
#define	EVENT_MESSAGE_RENDERING_SYSTEM_PRESENT_BEGIN	0x310
#define	EVENT_MESSAGE_RENDERING_SYSTEM_PRESENT_END	0x311
#define	EVENT_MESSAGE_CIGI_SESSION_STARTED	0x0400
#define	EVENT_MESSAGE_CIGI_SESSION_STOPPED	0x0401

Variables

REFIID **IID_IParameterListV400** = __uuidof(IParameterListV400)

REFIID **IID_ICallbackV400** = __uuidof(ICallbackV400)

REFIID	IID_IEventServiceV400 = __uuidof(IEventServiceV400)
REFIID	SID_EventService = __uuidof(IEventServiceV400)
GUID	EVENTID_Frame Event that is signaled once per frame. More...
GUID	EVENTID_1Hz This event is signaled once per frame. More...
GUID	EVENTID_CustomObjectDraw
GUID	EVENTID_Message

Macro Definition Documentation

◆ EVENT_MESSAGE_APPLICATION_ACTIVATED

```
#define EVENT_MESSAGE_APPLICATION_ACTIVATED 0x003b
```

Application just received WM_ACTIVATEAPP.

◆ EVENT_MESSAGE_APPLICATION_DEACTIVATED

```
#define EVENT_MESSAGE_APPLICATION_DEACTIVATED 0x003c
```

Application just received WM_ACTIVATEAPP.

◆ EVENT_MESSAGE_APPLICATION_DLLS_LOADED

```
#define EVENT_MESSAGE_APPLICATION_DLLS_LOADED 0x003e
```

Application dlls and plugin dlls are done loading.

◆ EVENT_MESSAGE_APPLICATION_RESTART

```
#define EVENT_MESSAGE_APPLICATION_RESTART 0x003a
```

Application is being "restarted" by a second instance.

◆ EVENT_MESSAGE_APPLICATION_SHUTDOWN

```
#define EVENT_MESSAGE_APPLICATION_SHUTDOWN 0x0037
```

Application is shutting down.

◆ EVENT_MESSAGE_APPLICATION_STARTUP

```
#define EVENT_MESSAGE_APPLICATION_STARTUP 0x0036
```

Application is starting up.

◆ EVENT_MESSAGE_AVATAR_ATTACH_CHANGED

```
#define EVENT_MESSAGE_AVATAR_ATTACH_CHANGED 0x280
```

The user's avatar has either attached or detached.

◆ EVENT_MESSAGE_CHANGE_USER_VEHICLE_CREATED

```
#define EVENT_MESSAGE_CHANGE_USER_VEHICLE_CREATED 0x241
```

New User Vehicle is created but hasn't been registered.

◆ EVENT_MESSAGE_CHANGE_USER_VEHICLE_FINISH

```
#define EVENT_MESSAGE_CHANGE_USER_VEHICLE_FINISH 0x242
```

User vehicle has finished changing.

◆ EVENT_MESSAGE_CHANGE_USER_VEHICLE_START

```
#define EVENT_MESSAGE_CHANGE_USER_VEHICLE_START 0x240
```

User vehicle is changing.

◆ EVENT_MESSAGE_CIGI_SESSION_STARTED

```
#define EVENT_MESSAGE_CIGI_SESSION_STARTED 0x0400
```

◆ EVENT_MESSAGE_CIGI_SESSION_STOPPED

```
#define EVENT_MESSAGE_CIGI_SESSION_STOPPED 0x0401
```

◆ EVENT_MESSAGE_CONTEXT_MENU_MODE_START

```
#define EVENT_MESSAGE_CONTEXT_MENU_MODE_START 0x0042
```

Context menu is getting ready to be displayed.

◆ EVENT_MESSAGE_CONTEXT_MENU_MODE_STOP

```
#define EVENT_MESSAGE_CONTEXT_MENU_MODE_STOP 0x0043
```

Context menu is done being displayed.

◆ EVENT_MESSAGE_CRASH_COMPLETE

```
#define EVENT_MESSAGE_CRASH_COMPLETE 0x0082
```

The user plane crashed and the crash sequence is done.

◆ EVENT_MESSAGE_CRASH_START

```
#define EVENT_MESSAGE_CRASH_START 0x0081
```

The user's plane just started crash sequence.

◆ EVENT_MESSAGE_CREATE_WINDOW

```
#define EVENT_MESSAGE_CREATE_WINDOW 0x0151
```

New window was created.

◆ EVENT_MESSAGE_CURRENT_SCENARIO_CHANGED

```
#define EVENT_MESSAGE_CURRENT_SCENARIO_CHANGED 0x002F
```

Global current scenario changed.

◆ EVENT_MESSAGE_DESTROY_WINDOW

```
#define EVENT_MESSAGE_DESTROY_WINDOW 0x0150
```

Window is about to be destroyed.

◆ EVENT_MESSAGE_DEVICE_RESET_COMPLETED

```
#define EVENT_MESSAGE_DEVICE_RESET_COMPLETED 0x0181
```

Rendering system finished remapping windows to devices.

◆ **EVENT_MESSAGE_DIALOG_MODE_START**

```
#define EVENT_MESSAGE_DIALOG_MODE_START 0x0044
```

A dialogue is getting ready to be displayed.

◆ **EVENT_MESSAGE_DIALOG_MODE_STOP**

```
#define EVENT_MESSAGE_DIALOG_MODE_STOP 0x0045
```

A dialogue is done being displayed.

◆ **EVENT_MESSAGE_ENABLE_VR_ON_STARTUP**

```
#define EVENT_MESSAGE_ENABLE_VR_ON_STARTUP 0x300
```

◆ **EVENT_MESSAGE_HMD_CONNECTED**

```
#define EVENT_MESSAGE_HMD_CONNECTED 0x01A3
```

Sent when an HMD is connected.

◆ **EVENT_MESSAGE_HMD_DISCONNECTED**

```
#define EVENT_MESSAGE_HMD_DISCONNECTED 0x01A4
```

Sent when an HMD is disconnected.

◆ **EVENT_MESSAGE_HMD_VIEW_CLOSED**

```
#define EVENT_MESSAGE_HMD_VIEW_CLOSED 0x01A6
```

Sent when an HMD view is turned off.

◆ **EVENT_MESSAGE_HMD_VIEW_DISABLE**

```
#define EVENT_MESSAGE_HMD_VIEW_DISABLE 0x01A7
```

Send to disable current HMD view and revert to previous.

◆ EVENT_MESSAGE_HMD_VIEW_OPENED

```
#define EVENT_MESSAGE_HMD_VIEW_OPENED 0x01A5
```

Sent when an HMD view is turned on.

◆ EVENT_MESSAGE_LOADING_COMPLETE

```
#define EVENT_MESSAGE_LOADING_COMPLETE 0x0197
```

Done loading.

◆ EVENT_MESSAGE_MAIN_LOADING_STATE_CHANGE

```
#define EVENT_MESSAGE_MAIN_LOADING_STATE_CHANGE 0x0191
```

On MainLoadingState change send new state.

◆ EVENT_MESSAGE_MAIN_WINDOW_CHANGE

```
#define EVENT_MESSAGE_MAIN_WINDOW_CHANGE 0x003d
```

Application main window maximized, etc.

◆ EVENT_MESSAGE_MEMORY_LOW

```
#define EVENT_MESSAGE_MEMORY_LOW 0x0046
```

A malloc has failed, if you see this free up some memory.

◆ EVENT_MESSAGE_MENU_MODE_START

```
#define EVENT_MESSAGE_MENU_MODE_START 0x0040
```

Main menu is getting ready to be displayed.

◆ EVENT_MESSAGE_MENU_MODE_STOP

```
#define EVENT_MESSAGE_MENU_MODE_STOP 0x0041
```

Main menu is done being displayed.

◆ EVENT_MESSAGE_MULTIPLAYER_CLIENT_STARTED

```
#define EVENT_MESSAGE_MULTIPLAYER_CLIENT_STARTED 0x0193
```

When a client has joined a session.

◆ EVENT_MESSAGE_MULTIPLAYER_LAUNCHED

```
#define EVENT_MESSAGE_MULTIPLAYER_LAUNCHED 0x0194
```

When the multiplayer session is launched.

◆ EVENT_MESSAGE_MULTIPLAYER_SERVER_STARTED

```
#define EVENT_MESSAGE_MULTIPLAYER_SERVER_STARTED 0x0192
```

When the host has created the server.

◆ EVENT_MESSAGE_MULTIPLAYER_SESSION_ENDED

```
#define EVENT_MESSAGE_MULTIPLAYER_SESSION_ENDED 0x0195
```

When the multiplayer session is ended.

◆ EVENT_MESSAGE_PANEL_CLOSED

```
#define EVENT_MESSAGE_PANEL_CLOSED 0x0232
```

Panel closed.

◆ EVENT_MESSAGE_PANEL_OPENED

```
#define EVENT_MESSAGE_PANEL_OPENED 0x0231
```

Panel opened.

◆ EVENT_MESSAGE_RENDERING_SYSTEM_PRESENT_BEGIN

```
#define EVENT_MESSAGE_RENDERING_SYSTEM_PRESENT_BEGIN 0x310
```

◆ EVENT_MESSAGE_RENDERING_SYSTEM_PRESENT_END

```
#define EVENT_MESSAGE_RENDERING_SYSTEM_PRESENT_END 0x311
```

◆ **EVENT_MESSAGE_REQUESTED_SIM_RATE_CHANGED**

```
#define EVENT_MESSAGE_REQUESTED_SIM_RATE_CHANGED 0x011A
```

Sends the new value of the sim speed.

◆ **EVENT_MESSAGE_SCENARIO_LOADED**

```
#define EVENT_MESSAGE_SCENARIO_LOADED 0x0028
```

Sent at end of scenario load sequence.

◆ **EVENT_MESSAGE_SCENARIO_LOADING**

```
#define EVENT_MESSAGE_SCENARIO_LOADING 0x0027
```

Sent at beginning of scenario load sequence.

◆ **EVENT_MESSAGE_SCENARIO_POST_LOADED**

```
#define EVENT_MESSAGE_SCENARIO_POST_LOADED 0x0030
```

After scenario PostLoad is completed.

◆ **EVENT_MESSAGE_SIM_FROZEN**

```
#define EVENT_MESSAGE_SIM_FROZEN 0x270
```

Sim frozen.

◆ **EVENT_MESSAGE_SIM_PAUSED**

```
#define EVENT_MESSAGE_SIM_PAUSED 0x0038
```

Simulation is paused.

◆ **EVENT_MESSAGE_SIM_UNFROZEN**

```
#define EVENT_MESSAGE_SIM_UNFROZEN 0x271
```

Sim unfrozen.

◆ EVENT_MESSAGE_SIM_UNPAUSED

```
#define EVENT_MESSAGE_SIM_UNPAUSED 0x0039
```

Simulation is unpause.

◆ EVENT_MESSAGE_SIMCONNECT_SIMULATION_START

```
#define EVENT_MESSAGE_SIMCONNECT_SIMULATION_START 0x0022
```

SimConnect has detected "simulating" has started.

◆ EVENT_MESSAGE_SIMCONNECT_SIMULATION_STOP

```
#define EVENT_MESSAGE_SIMCONNECT_SIMULATION_STOP 0x0023
```

SimConnect has detected "simulating" has stopped.

◆ EVENT_MESSAGE_SIMDIRECTOR_ACTIVE

```
#define EVENT_MESSAGE_SIMDIRECTOR_ACTIVE 0x022D
```

◆ EVENT_MESSAGE_SIMDIRECTOR_INACTIVE

```
#define EVENT_MESSAGE_SIMDIRECTOR_INACTIVE 0x022E
```

◆ EVENT_MESSAGE_TIMESEASON_CHANGED_10MIN

```
#define EVENT_MESSAGE_TIMESEASON_CHANGED_10MIN 0x0110
```

10 minute time change

◆ EVENT_MESSAGE_TIMESEASON_CHANGED_1MIN

```
#define EVENT_MESSAGE_TIMESEASON_CHANGED_1MIN 0x0112
```

1 minute time change

◆ EVENT_MESSAGE_TIMESEASON_CHANGED_BY_USER

```
#define EVENT_MESSAGE_TIMESEASON_CHANGED_BY_USER 0x0113
```

User has changed the time via the time/season dialogue.

◆ EVENT_MESSAGE_VIEW_ACTIVATE

```
#define EVENT_MESSAGE_VIEW_ACTIVATE 0x0160
```

Window activated.

◆ EVENT_MESSAGE_VIEW_CREATE

```
#define EVENT_MESSAGE_VIEW_CREATE 0x0163
```

Window created.

◆ EVENT_MESSAGE_VIEW_DESTROY

```
#define EVENT_MESSAGE_VIEW_DESTROY 0x0164
```

Window destroyed.

◆ EVENT_MESSAGE_VIEW_MODE_CHANGED

```
#define EVENT_MESSAGE_VIEW_MODE_CHANGED 0x0161
```

View changed on a window.

◆ EVENT_MESSAGE_WEATHER_CHANGED

```
#define EVENT_MESSAGE_WEATHER_CHANGED 0x0133
```

If there's any change in the weather, this message will get sent.

◆ EVENT_MESSAGE_WEATHER_MODE_CHANGED

```
#define EVENT_MESSAGE_WEATHER_MODE_CHANGED 0x0138
```

Weather system mode (i.e. RWW, theme, custom) has been changed.

Variable Documentation

◆ EVENTID_1Hz

GUID EVENTID_1Hz

This event is signaled once per frame.

◆ EVENTID_CustomObjectDraw

GUID EVENTID_CustomObjectDraw

This event is signaled once per view each frame. The [IPParameterListV400](#) can be used to request the IObjectRendererV400 service, which can be used to render custom objects into the scene. See the CustomLights example for more details.

(SDK\Utilities\PDK\CustomObjectRenderingSamples\CustomLights)

◆ EVENTID_Frame

GUID EVENTID_Frame

Event that is signaled once per frame.

◆ EVENTID_Message

GUID EVENTID_Message

Event that is fired whenever a Prepar3D message is sent. The EVENT_MESSAGE_message ID will be the first parameter of the EVENTID_Message callback. A sample message handler is included below:

```
virtual void Invoke(IP3DCallbackParams* pParams)
{
    UINT64 messageID = pParams->GetParameter(0).Value;
    if(messageID == EVENT_MESSAGE_APPLICATION_SHUTDOWN)
    {
        // handle app shutdown
    }
}
```

◆ IID_ICallbackV400

REFIID IID_ICallbackV400 = __uuidof(ICallbackV400)

◆ IID_IEventServiceV400

REFIID IID_IEventServiceV400 = __uuidof(IEventServiceV400)

◆ IID_IParameterListV400

REFIID IID_IParameterListV400 = __uuidof(**IParameterListV400**)

◆ SID_EventService

REFIID SID_EventService = __uuidof(**IEventServiceV400**)

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Global Data Service

Overview

This service provides certain global data that is not associated with any specific object in the application such as time and pause state. A sample implementation using this interface can be found in the SimpleAirplane sample in the ISimObject Samples.

Classes

class **IGlobalDataV430**

Class Documentation

◆ P3D::IGlobalDataV430

class P3D::IGlobalDataV430

Service for requesting general data not associated with any specific object in the application.

Remarks

Various "time" queries are made relative to either Zulu (UTC) time Local time using the TIMEREF enum.

Inherits IGlobalDataV400.

Private Member Functions

virtual **BOOL** **IsPaused** () const **PURE**

virtual float **GetSimRate** () const **PURE**

virtual **LICENSE_TYPE** **GetLicenseType** () const **PURE**

virtual HRESULT **GetUnitCode** (__in LPCWSTR pszPropertyUnits, __out int &iUnitCode) const **PURE**

virtual HRESULT **GetAbsoluteTime** (__out double &dValue, __in int eUnits) const

PURE	
virtual HRESULT	GetTime (_out double &dValue, _in int eUnits, _in TIMEREF eTimeRef) const PURE
virtual HRESULT	GetYear (_out UINT &uValue, _in TIMEREF eTimeRef) const PURE
virtual HRESULT	GetMonthOfTheYear (_out UINT &uValue, _in TIMEREF eTimeRef) const PURE
virtual HRESULT	GetDayOfTheYear (_out UINT &uValue, _in TIMEREF eTimeRef) const PURE
virtual HRESULT	GetDayOfTheMonth (_out UINT &uValue, _in TIMEREF eTimeRef) const PURE
virtual HRESULT	GetDayOfTheWeek (_out UINT &uValue, _in TIMEREF eTimeRef) const PURE
virtual HRESULT	GetTimeZoneOffset (_out double &dValue, _in int eUnits) const PURE
virtual TIMEofday	GetGeneralTimeOfDay () const PURE
virtual HRESULT	SetAbsoluteTime (_in double dValue, _in int eUnits) PURE
virtual HRESULT	SetTime (_in UINT uHour, _in UINT uMinute, _in UINT uSeconds, _in TIMEREF eTimeRef) PURE
virtual HRESULT	SetDate (_in UINT uYear, _in UINT uMonth, _in UINT uDayOfTheMonth, _in TIMEREF eTimeRef) PURE

Member Function Documentation

◆ GetAbsoluteTime()

```
virtual HRESULT GetAbsoluteTime ( _out double & dValue,
                                 _in int           eUnits
                               ) const    private virtual
```

Time since 00:00:00 year 0000

Parameters

[out] **dValue** Will contain requested value
eUnits Units of return value

◆ GetDayOfTheMonth()

```
virtual HRESULT GetDayOfTheMonth ( _out UINT & uValue,
                                   _in TIMEREF eTimeRef
                                 ) const    private virtual
```

Get day of the month

Parameters

[out] **uValue** Will contain requested value
eTimeRef Indicates local or zulu time

◆ GetDayOfTheWeek()

```
virtual HRESULT GetDayOfTheWeek ( __out UINT & uValue,  
                                 __in TIMEREF eTimeRef  
                               ) const private virtual
```

Get day of week

Parameters

[out] **uValue** Will contain requested value
eTimeRef Indicates local or zulu time

◆ GetDayOfTheYear()

```
virtual HRESULT GetDayOfTheYear ( __out UINT & uValue,  
                                 __in TIMEREF eTimeRef  
                               ) const private virtual
```

Get day of the year

Parameters

[out] **uValue** Will contain requested value
eTimeRef Indicates local or zulu time

◆ GetGeneralTimeOfDay()

```
virtual TIMEOFDAY GetGeneralTimeOfDay ( ) const private virtual
```

General time of day.

Returns

time of day.

◆ GetLicenseType()

```
virtual LICENSE_TYPE GetLicenseType ( ) const private virtual
```

License Type - Get application License Type

Returns

License type.

◆ GetMonthOfTheYear()

```
virtual HRESULT GetMonthOfTheYear ( __out UINT & uValue,  
                                  __in TIMEREF eTimeRef  
                                ) const private virtual
```

Get month of year.

Parameters

[out] **uValue** Will contain requested value
eTimeRef Indicates local or zulu time

◆ GetSimRate()

```
virtual float GetSimRate ( ) const private virtual
```

Sim Rate. Percentage scalar on real time. Typically ranges 0.25 - 128.0. Normal = 1.0.

Returns

The current simulation rate.

◆ GetTime()

```
virtual HRESULT GetTime ( __out double & dValue,  
                        __in int eUnits,  
                        __in TIMEREF eTimeRef  
                      ) const private virtual
```

Time of day (since midnight) Zulu or Local time ref

Parameters

[out] **dValue** Will contain requested value
eUnits Units of return value

◆ GetTimeZoneOffset()

```
virtual HRESULT GetTimeZoneOffset ( __out double & dValue,  
                                   __in int eUnits  
                                 ) const private virtual
```

Time Offset of current time zone from UTC

Parameters

[out] **dValue** Will contain requested value
eUnits Units of return value

◆ GetUnitCode()

```
virtual HRESULT GetUnitCode ( __in LPCWSTR pszPropertyUnits,
                            __out int & iUnitCode
                           )
                           const
                           private virtual
```

Provided here for convenience to convert unit string name to internal code

Parameters

pszPropertyUnits string name for unit
[out] **iUnitCode** Will contain requested value

◆ GetYear()

```
virtual HRESULT GetYear ( __out UINT & uValue,
                        __in TIMEREF eTimeRef
                       )
                       const
                       private virtual
```

Get year

Parameters

[out] **uValue** Will contain requested value
eTimeRef Indicates local or zulu time

◆ IsPaused()

```
virtual BOOL IsPaused ( ) const
                        private virtual
```

Paused - Is application paused

Returns

True if application is paused, false if not.

◆ SetAbsoluteTime()

```
virtual HRESULT SetAbsoluteTime ( __in double dValue,
                                 __in int eUnits
                                )
                                private virtual
```

Time since 00:00:00 year 0000

Parameters

dValue The requested time
eUnits Units of return value

◆ SetDate()

```
virtual HRESULT SetDate ( __in UINT uYear,
```

```
    __in UINT      uMonth,  
    __in UINT      uDayOfTheMonth,  
    __in TIMEREF  eTimeRef  
)
```

private virtual

Date in Zulu or Local time ref

Parameters

uYear The requested year
uMonth The requested month
uDayOfTheMonth The requested day of the month
eTimeRef Indicates local or zulu time

◆ SetTime()

```
virtual HRESULT SetTime ( __in UINT      uHour,  
                         __in UINT      uMinute,  
                         __in UINT      uSeconds,  
                         __in TIMEREF  eTimeRef  
)
```

private virtual

Time of day (since midnight) Zulu or Local time ref

Parameters

uHour The requested hour
uMinute The requested minute
uSeconds The requested seconds
eTimeRef Indicates local or zulu time

Variables

GUID IID_IGlobalDataV430

Current version of GlobalData interface. [More...](#)

GUID SID_GlobalData

GlobalData service ID. [More...](#)

Variable Documentation

◆ IID_IGlobalDataV430

GUID IID_IGlobalDataV430

Current version of GlobalData interface.

◆ SID_GlobalData

GUID SID_GlobalData

GlobalData service ID.

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Data Load Helper

Overview

This service is provided through the PDK interface and allows callers to load XML data using Prepar3D's own parser. The interface is defined in the file [IDataLoader.h](#). Data compiled into a binary XML files, known as .SPB files, can also be loaded with through this interface. A sample implementation using this interface can be found in the SimpleAirplane sample in the ISimObject Samples.

Classes

class [IDataLoadHelperV400](#)

Class Documentation

◆ [P3D::IDataLoadHelperV400](#)

class P3D::IDataLoadHelperV400

This interface allows requests for specific data in XML (or binary SPB) files. Requests are processed and returned through a callback provided by the caller for float numeric data and directly to a provided string pointer for string data. Returned data should be copied, as core Prepar3D will delete its copy upon return, and pointers to the data will become invalid.

Inherits [IUnknown](#).

Private Member Functions

```
virtual HRESULT LoadXmlFloatData (_in const void *pvID, _in __notnull const WCHAR *pszDataFile, _in __notnull const WCHAR *pszDataSetName, _in const WCHAR *pszDataSetSubName, _in __notnull PFloatLoadCallback pfCallback) const PURE
virtual HRESULT LoadXmlStringData (_in __notnull const WCHAR *pszDataFile, _in __notnull const WCHAR *pszDataSetName, _in const WCHAR *pszDataSetSubName, _out WCHAR *pszData, _in UINT uDataLength) const PURE
```

Member Function Documentation

◆ [LoadXmlFloatData\(\)](#)

```
virtual HRESULT
LoadXmlFloatData (_in const void * pvID,
                  _in __notnull const WCHAR * pszDataFile,
                  _in __notnull const WCHAR * pszDataSetName,
                  _in const WCHAR * pszDataSetSubName,
                  _in __notnull PFloatLoadCallback pfCallback
)
const private virtual
```

Reads a specified xml/spb, parses a list of floats into an array, and provides the list via callback function. The callback is synchronous. Return from this file occurs upon return from callback

Parameters

[in] pvID	ID passed in to identify the request in the callback.
[in] pszDataFile	File to read. If no extension, will try both SPB then XML

[in] **pszDataSetName** DataSet name
 [in] **pszDataSetSubName** Subname of specific element within the DataSet. If NULL, returns all subsets.
 [in] **pfCallback** Callback to receive data

Returns

S_OK if succeeded

Remarks

A callback is used here because the caller may not know the length of the return array, and the caller must allocate and maintain the required memory for the data.

◆ LoadXmlStringData()

```
virtual HRESULT
LoadXmlStringData( __in __notnull const WCHAR * pszDataFile,
                   __in __notnull const WCHAR * pszDataSetName,
                   __in const WCHAR * pszDataSetSubName,
                   __out WCHAR * pszData,
                   __in UINT uDataLength
)
const private virtual
```

Reads a specified xml/spb, and returns a string to a provided char pointer The callback is synchronous.

Parameters

[in] **pszDataFile** File to read. If no extension, will try both SPB then XML
 [in] **pszDataSetName** DataSet name
 [in] **pszDataSetSubName** Subname of specific element within the DataSet. If NULL, returns all subsets.
 [out] **pszData** Pointer to callers string
 [in] **uDataSize** Available string length

Returns

S_OK if succeeded

Sample XML Data

Note that if no extension is added to the file path passed into LoadXmlFloatData, Prepar3D will first attempt to load data from a ".SPB", and then fall back to a ".XML". The <FloatArray> data can be implemented as either an attribute or an element.

```
<SimBase.Document Type="AceXML" version="1,0">
  <Simvar.DataContainer Name = "Engine">
    <!-- Sample data 1 (single value as an attribute)-->
    <DataSet Name = "MaxRpm" FloatArray = "101.0"/>
    <!-- Sample data 2 (table data as an element)-->
    <DataSet Name = "EngineData">
      <FloatArray>
        <!-- in, out -->
        0.0, 0.2
        0.5, 1.0
      </FloatArray>
    </DataSet>
    <!-- Sample string data -->
    <DataSet Name = "Engine Name" String = "Left Engine"/>
  </Simvar.DataContainer>
</SimBase.Document>
```

TypeDefs

```
typedef HRESULT(STDMETHODCALLTYPE * PFloatLoadCallback) ( __in const void *pID, __in
                           const WCHAR *pszDataSetName, __in const
                           WCHAR *pszDataSetSubName, __in const float
                           *rgFloat, __in UINT nFloats)
```

Variables

GUID IID_IDataLoadHelperV400
GUID SID_DataLoadHelper

Typedef Documentation

◆ PFloatLoadCallback

```
typedef HRESULT(STDMETHODCALLTYPE * PFloatLoadCallback) (_in const void *pID, _in  
const WCHAR *pszDataSetName, _in const WCHAR *pszDataSetSubName, _in const float  
*rgFloat, _in UINT nFloats)
```

Callback function type used for loading XML or SPB file data through the `IDataLoadHelper` interface. This is the callback type that will receive the float data. The pointer for the ID and string names passed into `LoadXmlFloatData` are passed through to this callback for easy comparison. The data load request will be made with a DataSet name and sub-name for an element within the DataSet. These two names are passed back through the callback for reference, along with an array pointer and number of floats.

The void pointer can be used to identify an object or ID from the caller. The callback is synchronous and the array will be destroyed upon return from the callback.

Variable Documentation

◆ IID_IDataLoadHelperV400

GUID IID_IDataLoadHelperV400

◆ SID_DataLoadHelper

GUID SID_DataLoadHelper

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Air-To-Ground Radar Service

Overview

Prepar3D provides an air-to-ground radar simulation and visualization service for developers. Because every radar system is different, the Prepar3D radar is provided as a highly configurable service which can be controlled via C++ plug-ins and XML gauges. Some examples of controllable parameters are: Range, Sweep Angle, Sweep Rate, Zoom Level, Image, and Data resolution. Some of the advanced capabilities of the Prepar3D radar include accurate radar shadows, far-shore-enhancement, and Doppler-beam-sharpening. Also, we developed the radar service entirely through our SDK to showcase the power and flexibility that 3rd party developers have.

Radar XML Interface

The quickest and simplest way to get up and running using the Prepar3D radar is to write an XML gauge using the C:P3DRadar variables provided by the P3DRadar IPanelCallback. This panel callback is loaded automatically, so there is no need to add a dll to the DLL.XML. A sample gauge that uses most of the provided functionality is included in the core application gauges folder (gauges\P3DRadar\radar.xml). To add Prepar3D's sample radar gauge to a SimObject, simply open up the SimObject's panel.cfg and add a panel entry that references the P3DRadar:radar gauge:

```
[Window Titles]
Window00=Radar
[Window00]
size_mm=265,310
Background_color=1,1,1
visible=0
position=8
windowsize_ratio=0.4
gauge00=P3DRadar!Radar, 5, 5, 255, 305
```

To extend the example or write new XML gauges based on this interface, refer to the full list of C:P3DRadar variables below. P3DRadar C Variable Description Units Settable Multiplayer

P3DRadar C Variable	Description	Units	Settable	Multiplayer
ClearRadarImage	Set non zero value to trigger a radar image clear. 1 - Show range rings 0 -	Number	Y	N

ShowRangeRings	1 - Show range rings 0 - Hide range rings	Number	Y	N
ShowCursor	1 - Show Cursor 0 - Hide Cursor	Number	Y	N
FarShoreEnhance	1 - Enable Far Shore Enhancement 0 - Disable Far Shore Enhancement	Number	Y	N
VisualZoom	Zoom in on radar display. Display is centered on current cursor position. Values less than 1.0 are not supported.	Number	Y	N
DataZoom	Remap radar sensor coverage to area surrounding the current cursor position to increase data resolution. When set to the same value as visual zoom, this works well as an approximation of Doppler Beam Sharpening. Values less than 1.0 are not supported.	Number	Y	N
ScanAzimuth	Maximum deviation from center of the radar system sweep. Valid Range is 1.0-60.0.	Number	Y	N
SweepRate	Sweep rate in degrees per second	Number	Y	N
RangeMiles	Maximum range of radar in Nautical Miles.	Number	Y	N
RenderingEnabled	The effectively disables the radar service. This should be set to 0 when not displaying the radar texture element with the radar for best performance. 1 - Enable rendering 0 - Disable rendering	Number	Y	N
FreezeEnabled	1 - Enable Image and Radar Beam Freeze 0 - Disable Image and Radar Beam Freeze	Number	Y	N
CursorPositionX	Set cursor horizontal cursor location. The range 0.0-1.0 maps from left to right across the extents of the un-zoomed radar gauge image. Zoom calculation must be done by developer.	Number	Y	N
	Set cursor vertical cursor location. The range 0.0-			

CursorPositionY	1.0 maps from top to bottom down the extents of the un-zoomed radar gauge image. Zoom calculation must be done by developer.	Number	Y	N
CursorPositionLat	Latitude of cursor position. Actual cursor position in radar is relative to display. Getting or setting this value will trigger conversions in the C++. For best results set Latitude and Longitude in pairs.	Number	Y	N
CursorPositionLon	Longitude of cursor position. Actual cursor position in radar is relative to display. Getting or setting this value will trigger conversions in the C++. For best results set Latitude and Longitude in pairs.	Number	Y	N
FrontBlindSpotDegrees	Width of the current front blindspot in degrees. This is generally only used for Doppler Beam Sharpening and should otherwise be set to 0.	Number	Y	N
SideBlindSpotDegrees	Width of the current side blindspot in degrees. This is generally only used for Doppler Beam Sharpening and should otherwise be set to 0.	Number	Y	N
RadarResolutionX	Horizontal resolution of radar data texture.	Number	Y	N
RadarResolutionY	Vertical resolution of radar data texture.	Number	Y	N
GaugeResolutionX	Horizontal resolution of radar gauge texture in pixels.	Number	Y	N
GaugeResolutionY	Vertical resolution of radar gauge texture in pixels.	Number	Y	N
CurrentRadarScanElevationDegrees	Current elevation of the radar beam in degrees. Read-only value set by radar system. This value will vary by altitude, range, and data zoom.	Number	N	N
	Get the current angular offset of the radar beam.			

CurrentRadarBeamOffset	Read-only value determined by radar simulation.	Number	N	N
------------------------	---	--------	---	---

Radar Panel Callback Example

For more advanced use cases, the Prepar3D radar can be controlled via C++ using the ISimulatedRadar PDK Service. One possible use case for this would be to provide a customized IPanelCallback that extends the capabilities of the P3DRadar callback provided by Prepar3D. The **Radar** Callback Sample is a great starting point for this use case and can be found here:

<Prepar3D SDK Path>\PDK\RadarPanelCallbackSample

The **Radar** Callback Sample source can be viewed [here](#). The **Radar** Callback Sample is a mixed mode gauge example similar to the CabinComfort example but it is far more advanced. The PanelCallback in the sample is functionally equivalent to the PanelCallback provided by the core radar service, so it is a perfect starting point for extending the existing XML interface. Controlling the radar from C++ provides more flexibility and higher performance, and can greatly simplify the XML scripting required to create a robust radar gauge. Interfacing via C++ also can allow multiple add-ons to interface with the radar which may be required to simulate more advanced aircraft. Consult the ReadMe file in the example folder for further instructions on how to install and use it.

Classes

class **ISimulatedRadarV440**

Class Documentation

◆ Radar::ISimulatedRadarV440

class Radar::ISimulatedRadarV440

Interface for **Radar** PDK service

Inherits ISimulatedRadarV430.

Public Member Functions

virtual HRESULT **Init** (const WCHAR *sRenderToTextureName, UINT width, UINT height)
override

virtual HRESULT **DeInit** (void) override

 virtual bool **IsInitialized** () const override

 virtual bool **NeedsUpdate** () const override

 virtual bool **ShowRangeRings** () const override

 virtual bool **ShowCursor** () const override

 virtual bool **FarShoreEnhance** () const override

 virtual double **GetVisualZoom** () const override

 virtual double **GetDataZoom** () const override

 virtual double **GetScanAzimuth** () const override

	<code>virtual double GetScanAzimuth() const override</code>
	<code>virtual double GetSweepRate() const override</code>
	<code>virtual double GetRangeMiles() const override</code>
	<code>virtual bool RenderingEnabled() const override</code>
	<code>virtual bool FreezeEnabled() const override</code>
	<code>virtual void GetCursorPositionXY(double &x, double &y) const override</code>
	<code>virtual void GetCursorPositionLLA(LLA &lla) const override</code>
	<code>virtual double GetFrontBlindspotDegrees() const override</code>
	<code>virtual double GetSideBlindspotDegrees() const override</code>
	<code>virtual void GetRadarResolution(double &x, double &y) const override</code>
	<code>virtual void GetGaugeResolution(double &x, double &y) const override</code>
	<code>virtual double GetCurrentRadarScanElevationDegrees() const override</code>
	<code>virtual double GetCurrentRadarBeamOffsetDegrees() const override</code>
	<code>virtual void ClearRadarImage() override</code>
	<code>virtual void SetShowRangeRings(bool show) override</code>
	<code>virtual void SetShowCursor(bool enabled) override</code>
	<code>virtual void SetFarShoreEnhancementEnabled(bool enabled) override</code>
	<code>virtual void SetVisualZoom(double visualZoom) override</code>
	<code>virtual void SetDataZoom(double dataZoom) override</code>
	<code>virtual void SetScanRateDegreesPerSecond(double rate) override</code>
	<code>virtual void SetRangeMiles(double range) override</code>
	<code>virtual void SetRenderingEnabled(bool enabled) override</code>
	<code>virtual void SetFreeze(bool freeze) override</code>
	<code>virtual void SetCursorPositionXY(double dX, double dY) override</code>
	<code>virtual void SetCursorPositionLLA(const LLA &lla) override</code>
	<code>virtual void SetRadarImageResolution(double resolutionX, double resolutionY) override</code>
	<code>virtual void SetRadarGaugeResolution(double resolutionX, double resolutionY) override</code>
	<code>virtual void SetScanAzimuthDegrees(double azimuth) override</code>
	<code>virtual void SetFrontBlindSpotDegrees(double frontBlindSpot) override</code>
	<code>virtual void SetSideBlindSpotDegrees(double sideBlindSpots) override</code>
	<code>virtual void SetRadarScanColor(float red, float green, float blue) override</code>
	<code>virtual void SetRadarGain(float gain) override</code>
	<code>virtual void SetRadarTextureInterpolation(bool interpolate) override</code>
	<code>virtual void UseCustomRadarAngles(bool useCustomRadarAntenna) override</code>
	<code>virtual void SetCustomRadarAngles(double antennaElevation, double antennaFov) override</code>
	<code>virtual void GetLLAFromLocalXY(LLA &LLA, double dX, double dY) const override</code>
	<code>virtual void GetLLAFromGlobalXY(LLA &LLA, double dX, double dY) const override</code>
	<code>virtual void GetLocalXYCoord(double &dX, double &dY, const LLA &LLA) const override</code>

```
virtual void GetGlobalXYCoord (double &dX, double &dY, const LLA &LLA) const  
override
```

Member Function Documentation

◆ **ClearRadarImage()**

```
virtual void ClearRadarImage ( ) pure virtual
```

Clear the radar image on update

◆ **DeInit()**

```
virtual HRESULT DeInit ( void ) pure virtual
```

Deinit the radar system

◆ **FarShoreEnhance()**

```
virtual bool FarShoreEnhance ( ) const pure virtual
```

Is the radar image enhancing far shore pixels

Returns

Far shore enhancement enabled

◆ **FreezeEnabled()**

```
virtual bool FreezeEnabled ( ) const pure virtual
```

Is the radar beam actively sweeping and updating its intermediate texture? Note: When in freeze mode, the radar camera and intermediate texture compositing is disabled, but the gauge is still rendered

Returns

Radar is actively sweeping

◆ **GetCurrentRadarBeamOffsetDegrees()**

```
virtual double GetCurrentRadarBeamOffsetDegrees ( ) const pure virtual
```

Get the current angular offset of the radar beam

Returns

Beam offset in degrees

◆ GetCurrentRadarScanElevationDegrees()

```
virtual double GetCurrentRadarScanElevationDegrees( ) const pure virtual
```

Get the current elevation of the radar beam in degrees

Returns

Elevation of radar beam

◆ GetCursorPositionLLA()

```
virtual void GetCursorPositionLLA ( LLA & lla ) const pure virtual
```

Get the **LLA** of the current cursor position

Parameters

lla Out variable for the current **LLA** of the radar cursor

◆ GetCursorPositionXY()

```
virtual void GetCursorPositionXY ( double & x,  
                                 double & y  
                               ) const pure virtual
```

Get the current x and y coordinates of the cursor

Parameters

x Out variable for X coordinate of cursor

y Out variable for y coordinate of cursor

◆ GetDataZoom()

```
virtual double GetDataZoom( ) const pure virtual
```

Get the current level of data zoom

Returns

Data zoom

◆ GetFrontBlindspotDegrees()

```
virtual double GetFrontBlindspotDegrees( ) const pure virtual
```

Get the width of the current front blind spot in degrees This will usually be 0 unless the radar is in a DBS mode

Returns

Width of front blind spot in degrees

◆ GetGaugeResolution()

```
virtual void GetGaugeResolution ( double & x,  
                                double & y  
                            ) const pure virtual
```

Get the resolution of the radar gauge

Parameters

x Width of radar gauge
y Height of radar gauge

◆ GetGlobalXYCoord()

```
virtual void GetGlobalXYCoord ( double & dX,  
                               double & dY,  
                               const LLA & LLA  
                           ) const pure virtual
```

Get **XY** position of an **LLA** relative to un-zoomed radar display. X and Y are 0-1, top-left to bottom-right of un-zoomed radar display.

Parameters

dX[out] Local x coordinate relative to un-zoomed radar display
dY[out] Local y coordinate relative to un-zoomed radar display
LLA Global **LLA** position to calculated

◆ GetLLAFromGlobalXY()

```
virtual void GetLLAFromGlobalXY ( LLA & LLA,  
                                 double dX,  
                                 double dY  
                             ) const pure virtual
```

Get **LLA** of a point on the radar display. X and Y are 0-1, top-left to bottom-right of un-zoomed radar display.

Parameters

LLA[out] Global **LLA** position to calculated
dX Local x coordinate relative to un-zoomed radar display
dY Local y coordinate relative to un-zoomed radar display

◆ GetLLAFromLocalXY()

```
virtual void GetLLAFromLocalXY ( LLA & LLA,  
                               double dX,  
                               double dY  
                           ) const pure virtual
```

Get **LLA** of a point on the radar display. X and Y are 0-1, top-left to bottom-right of radar display.

Parameters

LLA[out] Global **LLA** position to calculated
dX Local x coordinate relative to radar display
dY Local y coordinate relative to radar display

◆ GetLocalXYCoord()

```
virtual void GetLocalXYCoord ( double & dX,  
                             double & dY,  
                             const LLA & LLA  
                           ) const pure virtual
```

Get **XY** position of an **LLA** relative to zoomed radar display. X and Y are 0-1, top-left to bottom-right of zoomed radar display.

Parameters

dX[out] Local x coordinate relative to zoomed radar display
dY[out] Local y coordinate relative to zoomed radar display
LLA Global **LLA** position to calculated

◆ GetRadarResolution()

```
virtual void GetRadarResolution ( double & x,  
                                 double & y  
                               ) const pure virtual
```

Get the resolution of the radar image

Parameters

x Width of radar image
y Height of radar image

◆ GetRangeMiles()

```
virtual double GetRangeMiles ( ) const pure virtual
```

Get the current range setting of the radar system in miles

Returns

Radar range setting, in miles

◆ GetScanAzimuth()

virtual double GetScanAzimuth() const **pure virtual**

Get the maximum deviation from center of the radar system sweep

Returns

Sweep azimuth

◆ GetSideBlindspotDegrees()

virtual double GetSideBlindspotDegrees() const **pure virtual**

Get the width of the current side blind spots in degrees This will usually be 0 unless the radar is in a DBS mode This width is measured from the maximum extents of the current azimuth

Returns

Side blind spot width in degrees

◆ GetSweepRate()

virtual double GetSweepRate() const **pure virtual**

Get the rate at which the radar system sweeps side to side

Returns

Sweep rate in degrees per second

◆ GetVisualZoom()

virtual double GetVisualZoom() const **pure virtual**

Get the current visual zoom level

Returns

Visual zoom

◆ Init()

virtual HRESULT Init(const WCHAR * sRenderToTextureName,
 UINT width,
 UINT height
)

pure virtual

Initialize the radar system with the specified RTT name, and resolution

Parameters

sRenderTargetName	Name of RTT texture
width	Initial width for radar image and gauge
height	Initial height for radar image and gauge

◆ **IsInitialized()**

```
virtual bool IsInitialized( ) const pure virtual
```

Returns true if radar system has been initialized.

◆ **NeedsUpdate()**

```
virtual bool NeedsUpdate( ) const pure virtual
```

Since gauges might outpace framerate, gauge code should ensure that the radar System is not in need of an update before re-doing any work.

◆ **RenderingEnabled()**

```
virtual bool RenderingEnabled( ) const pure virtual
```

Is the radar currently rendering to the intermediate and final gauge texture? Note: the radar camera is still updated, along with radar variables

Returns

Internal gauge rendering enabled

◆ **SetCursorPositionLLA()**

```
virtual void SetCursorPositionLLA( const LLA & lla ) pure virtual
```

Set the cursor over the specified Lat Lon Alt

Parameters

lla New latlonalt of cursor, note that this must fall within radar picture

◆ **SetCursorPositionXY()**

```
virtual void SetCursorPositionXY( double dX,  
                               double dY  
                           ) pure virtual
```

Set the cursor position with local x and y coordinates X and Y are 0-1, top-left to bottom-right of radar display.

Parameters

dX Local x coordinate

dY Local y coordinate

◆ SetCustomRadarAngles()

```
virtual void SetCustomRadarAngles ( double antennaElevation,  
                                   double antennaFov  
                               )
```

pure virtual

Set the gauge texture to use a interpolation

Parameters

antennaElevation double declination angle in degrees below aircraft horizon

antennaFov double FOV angle in degrees centered about the antennaElevation

◆ SetDataZoom()

```
virtual void SetDataZoom ( double dataZoom )
```

pure virtual

Set the data zoom level Increasing visual zoom will improve resolution by reducing the area covered

Parameters

dataZoom Data zoom level

◆ SetFarShoreEnhancementEnabled()

```
virtual void SetFarShoreEnhancementEnabled ( bool enabled )
```

pure virtual

Set far shore enhancement enabled

Parameters

enabled Enable far shore enhancement

◆ SetFreeze()

```
virtual void SetFreeze ( bool freeze )
```

pure virtual

Set whether the radar is updating the sweep of the beam and intermediate texture

Parameters

freeze Radar freeze mode enabled

◆ SetFrontBlindSpotDegrees()

```
virtual void SetFrontBlindSpotDegrees ( double frontBlindSpot ) pure virtual
```

Set the front DBS blind spot width

Parameters

frontBlindSpot New width of front blind spot in degrees

◆ **SetRadarGain()**

```
virtual void SetRadarGain ( float gain ) pure virtual
```

Set the gain of the radar scan

Parameters

gainOffset New gain value -1.0 - 1.0

◆ **SetRadarGaugeResolution()**

```
virtual void SetRadarGaugeResolution ( double resolutionX,  
double resolutionY  
)  
pure virtual
```

Set the resolution of the gauge being rendered into. If the gauge resolution is set incorrectly the radar image will not display properly.

Parameters

resolutionX New width for gauge image

resolutionY New height for gauge image

◆ **SetRadarImageResolution()**

```
virtual void SetRadarImageResolution ( double resolutionX,  
double resolutionY  
)  
pure virtual
```

Set the resolution of the radar's internal texture.

Parameters

resolutionX New width for radar image

resolutionY New height for radar image

◆ **SetRadarScanColor()**

```
virtual void SetRadarScanColor ( float red,  
float green,  
float blue
```

)

pure virtual

Set the base color of the radar scan

Parameters

red New red value 0.0 - 1.0
green New green value 0.0 - 1.0
blue New blue value 0.0 - 1.0

◆ **SetRadarTextureInterpolation()**

virtual void SetRadarTextureInterpolation (bool **interpolate**) pure virtual

Set the gauge texture to use a interpolation

Parameters

interpolate bool to turn on/off radar texture interpolation

◆ **SetRangeMiles()**

virtual void SetRangeMiles (double **range**) pure virtual

Set the range of the radar system in miles

Parameters

range New radar rang in miles

◆ **SetRenderingEnabled()**

virtual void SetRenderingEnabled (bool **enabled**) pure virtual

Set whether the radar system is updating its intermediate texture and the final gauge image

Parameters

enabled Radar rendering enabled

◆ **SetScanAzimuthDegrees()**

virtual void SetScanAzimuthDegrees (double **azimuth**) pure virtual

Set the maximum extent of the radar sweep

Parameters

azimuth New azimuth in degrees

◆ **SetScanRateDegreesPerSecond()**

```
virtual void SetScanRateDegreesPerSecond ( double rate ) pure virtual
```

Set the scan rate of the radar beam

Parameters

rate New sweep rate

◆ **SetShowCursor()**

```
virtual void SetShowCursor ( bool enabled ) pure virtual
```

Set cursor drawing enabled

Parameters

enabled Enable simple cursor display

◆ **SetShowRangeRings()**

```
virtual void SetShowRangeRings ( bool show ) pure virtual
```

Set range rings enabled

Parameters

show Enable range ring display

◆ **SetSideBlindSpotDegrees()**

```
virtual void SetSideBlindSpotDegrees ( double sideBlindSpots ) pure virtual
```

Set the width of the side DBS blind spots

Parameters

sideBlindSpots New width of side blind spots

◆ **SetVisualZoom()**

```
virtual void SetVisualZoom ( double visualZoom ) pure virtual
```

Set the visual zoom level Increasing visual zoom is equivalent to zooming on a fixed resolution image

Parameters

visualZoom Visual zoom level

◆ ShowCursor()

virtual bool ShowCursor() const **pure virtual**

Is cursor drawing enabled?

Returns

Cursor enabled

◆ ShowRangeRings()

virtual bool ShowRangeRings() const **pure virtual**

Is range-ring display enabled?

Returns

Range-rings enabled

◆ UseCustomRadarAngles()

virtual void UseCustomRadarAngles(bool **useCustomRadarAntenna**) **pure virtual**

Set the radar to use the custom radar angles

Parameters

useCustomRadarAntenna bool to turn on/off using the custom radar angles

Variables

REFIID **IID_ISimulatedRadarV440** = __uuidof(**ISimulatedRadarV440**)

REFGUID **SID_SimulatedRadar** = __uuidof(**ISimulatedRadarV440**)

Variable Documentation

◆ IID_ISimulatedRadarV440

REFIID IID_ISimulatedRadarV440 = __uuidof(**ISimulatedRadarV440**)

◆ SID_SimulatedRadar

REFGUID SID_SimulatedRadar = __uuidof(**ISimulatedRadarV440**)

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Panel System Service

Overview

The IPanelSystem interface provides functionality and services previously only provided through the ImportTable in `gauges.h`. This service alleviates the need to declare the ImportTable and associated linkage.

The initial version mimics the functions specified in the `gauges.h` ImportTable. However, it changes the naming convention to a Camel Case Notation to differentiate the names. For example: `function_sample()` would be changed to `FunctionSample()`. Functionality remains the same.

An `IPdk` pointer can be obtained from the `IPdk QueryService()` method.

Classes

```
class IGaugeCDrawableCreateParameters  
class IGaugeCDrawableDrawParameters  
class IGaugeCDrawable  
class IGaugeCCallback  
class ISerializableGaugeCCallback  
class IAircraftCCallback  
class IPanelCCallback  
class IFSXPanelCCallback  
class IPanelSystemV400
```

Class Documentation

◆ IGaugeCDrawableCreateParameters

class IGaugeCDrawableCreateParameters

The `IGaugeCDrawableCreateParameters` class passes parameters into Prepar3D.

See the `Custom_Nav_Draw` sample for an example use case.

Public Types

```
enum PARAMETER_TYPE {  
    PARAMETER_TYPE_NONE = 0, PARAMETER_TYPE_BOOL = 1,  
    PARAMETER_TYPE_FLOAT = 2, PARAMETER_TYPE_INT = 3,  
    PARAMETER_TYPE_STRING = 4  
}
```

Public Member Functions

```
virtual bool GetParameter (const char *szParName, const char **pszValue) const =0  
virtual bool GetParameter (const char *szParName, FLOAT64 *pdValue) const =0  
virtual bool GetParameter (const char *szParName, INT_PTR *piValue) const =0  
virtual bool GetParameter (const char *szParName, bool *pbValue) const =0  
virtual bool SetParameterId (const char *szParName, SINT32 id, PARAMETER_TYPE type)  
const =0
```

Member Enumeration Documentation

◆ PARAMETER_TYPE

enum PARAMETER_TYPE

Enumerator
PARAMETER_TYPE_NONE
PARAMETER_TYPE_BOOL
PARAMETER_TYPE_FLOAT
PARAMETER_TYPE_INT
PARAMETER_TYPE_STRING

Member Function Documentation

◆ GetParameter() [1 / 4]

```
virtual bool GetParameter (const char * szParName,  
                           const char ** pszValue  
                           ) const pure virtual
```

Pass the `char*` value by finding the corresponding parameter name.

Parameters

szParName The parameter name.
pszValue The parameter's char* value.

Returns
True if parameter name and value found, false otherwise.

◆ **GetParameter()** [2/4]

```
virtual bool GetParameter ( const char * szParName,
    FLOAT64 * pdValue
) const pure virtual
```

Pass the FLOAT64 value by finding the corresponding parameter name.

Parameters
szParName The parameter name.
pdValue The parameter's FLOAT64 value.

Returns
True if parameter name and value found, false otherwise.

◆ **GetParameter()** [3/4]

```
virtual bool GetParameter ( const char * szParName,
    INT_PTR * piValue
) const pure virtual
```

Pass the SINT32 value by finding the corresponding parameter name.

Parameters
szParName The parameter name.
piValue The parameter's INT_PTR* value.

Returns
True if parameter name and value found, false otherwise.

◆ **GetParameter()** [4/4]

```
virtual bool GetParameter ( const char * szParName,
    bool * pbValue
) const pure virtual
```

Pass the bool value by finding the corresponding parameter name.

Parameters
szParName The parameter name.
pbValue The parameter's bool value.

Returns
True if parameter name and value found, false otherwise.

◆ **SetParameterId()**

```
virtual bool SetParameterId ( const char * szParName,
    SINT32 id,
    PARAMETER_TYPE type
) const pure virtual
```

Find the parameters name and sets the id and type that are passed in.

Parameters
szParName The parameter name.
id The parameter id.
type The parameter type.

Returns
True if szParName is found and other parameters are set, false otherwise.

◆ **IGaugeCDrawableDrawParameters**

class **IGaugeCDrawableDrawParameters**

The **IGaugeCDrawableDrawParameters** class gets the parameters passed into Prepar3D from the **IGaugeCDrawableCreateParameters** class.

See the Custom_Nav_Draw sample for an example use case.

Public Member Functions

virtual bool	GetParameter (SINT32 id, const char **pszValue) const =0
virtual bool	GetParameter (SINT32 id, FLOAT64 *pdValue) const =0
virtual bool	GetParameter (SINT32 id, INT_PTR *piValue) const =0
virtual bool	GetParameter (SINT32 id, bool *pbValue) const =0
virtual FLOAT64	GetScaleX () const =0
virtual FLOAT64	GetScaleY () const =0
virtual const PIXRECT *	GetScreenRectangle () const =0

Member Function Documentation

◆ **GetParameter()** [1/4]

```
virtual bool GetParameter( SINT32 id,  
                           const char ** pszValue  
                         ) const pure virtual
```

Passes the char* value by finding the corresponding id.

Parameters

id The parameter id.
pszValue The parameter's char* value.

Returns

True if parameter id and value found, false otherwise.

◆ **GetParameter()** [2 / 4]

```
virtual bool GetParameter( SINT32 id,  
                           FLOAT64 * pdValue  
                         ) const pure virtual
```

Passes the FLOAT64 value by finding the corresponding id.

Parameters

id The parameter id.
pdValue The parameter's FLOAT64 value.

Returns

True if parameter id and value found, false otherwise.

◆ **GetParameter()** [3 / 4]

```
virtual bool GetParameter( SINT32 id,  
                           INT_PTR * piValue  
                         ) const pure virtual
```

Passes the SINT32 value by finding the corresponding id.

Parameters

id The parameter id.
piValue The parameter's INT_PTR* value.

Returns

True if parameter id and value found, false otherwise.

◆ **GetParameter()** [4 / 4]

```
virtual bool GetParameter( SINT32 id,  
                           bool * pbValue  
                         ) const pure virtual
```

Passes the bool value by finding the corresponding id.

Parameters

id The parameter id.
pbValue The parameter's bool value.

Returns

True if parameter id and value found, false otherwise.

◆ **GetScaleX()**

```
virtual FLOAT64 GetScaleX( ) const pure virtual
```

Gets the scale of X.

Returns

Value of scale X.

◆ **GetScaleY()**

```
virtual FLOAT64 GetScaleY( ) const pure virtual
```

Gets the scale of Y.

Returns

Value of scale Y.

◆ **GetScreenRectangle()**

```
virtual const PIXRECT* GetScreenRectangle( ) const pure virtual
```

Gets screen rectangle.

Returns

Pointer to a rect relative to the flight sim window left top corner or NULL if undocked or virtual cockpit.

◆ **IGaugeCDrawable**

```
class IGaugeCDrawable
```

The **IGaugeCDrawable** class handles drawing the parameters entered.

See the Custom_Nav_Draw sample for an example use case.

Public Types

```
enum GAUGE_DRAWABLE_FLAGS {
    TAKES_DC = 0x1, TAKES_PIMAGE = 0x2, NOT_RESIZABLE = 0x4,
    DRAWS_ALPHA = 0x8,
    NO_TRANSPARENCY = 0x10, MASK_TRANSPARENCY = 0x20,
    DOUBLE_BUFFER = 0x40
}
```

Public Member Functions

```
virtual ULONG AddRef ()=0
virtual ULONG Release ()=0
virtual FLAGS32 GetFlags ()=0
virtual void Update ()=0
virtual void Show (bool on)=0
virtual bool Draw (IGaugeCDrawableDrawParameters *pParameters, PIXPOINT size,
                  HDC hdc, PIMAGE pImage)=0
virtual bool SetupDraw (PIXPOINT size, HDC hdc, PIMAGE pImage)=0
virtual bool GetDraw (IGaugeCDrawableDrawParameters *pParameters)=0
```

Member Enumeration Documentation

◆ GAUGE_DRAWABLE_FLAGS

enum GAUGE_DRAWABLE_FLAGS

Enumerator
TAKES_DC
TAKES_PIMAGE
NOT_RESIZABLE
DRAWS_ALPHA
NO_TRANSPARENCY
MASK_TRANSPARENCY
DOUBLE_BUFFER

Member Function Documentation

◆ AddRef()

```
virtual ULONG AddRef ( ) pure virtual
```

Add a reference.

◆ Draw()

```
virtual bool Draw (IGaugeCDrawableDrawParameters * pParameters,
                  PIXPOINT size,
                  HDC hdc,
                  PIMAGE pImage)
)
```

pure virtual

Draw is called if DOUBLE_BUFFER is disabled.

Parameters

pParameters Passes the parameters defined in the xml file.
size Size of the image.
hdc Pass through the hdc if TAKES_DC is enabled.
pImage Pass through the image if TAKES_PIMAGE is enabled.

Returns

If the image changed, return true for draw. False otherwise.

◆ GetDraw()

```
virtual bool GetDraw (IGaugeCDrawableDrawParameters * pParameters) pure virtual
```

GetDraw is called if DOUBLE_BUFFER is enabled.

Parameters

pParameters Passes the parameters defined in the xml file.

Returns

If the image changed, return true for draw. False otherwise.

◆ GetFlags()

```
virtual FLAGS32 GetFlags ( ) pure virtual
```

Gets the GAUGE_DRAWABLE_FLAGS which determine how this gauge drawable will be used.

Returns

Return a set of flags. Valid flags are listed in GAUGE_DRAWABLE_FLAGS.

Remarks

The flags tell Prepar3D how to setup the custom gauge element.

◆ **Release()**

```
virtual ULONG Release( ) pure virtual
```

Release the reference.

◆ **SetupDraw()**

```
virtual bool SetupDraw ( PIXPOINT size,
                        HDC      hdc,
                        PIMAGE   plImage
                      )

```

pure virtual

SetupDraw is called if DOUBLE_BUFFER is enabled.

Parameters

- size** Size of the image.
- hdc** Pass through the hdc if TAKES_DC is enabled.
- plImage** Pass through the image if TAKES_PIMAGE is enabled.

Returns

If the image changed, return true for draw. False otherwise.

◆ **Show()**

```
virtual void Show ( bool on ) pure virtual
```

Whether or not draw is visible.

◆ **Update()**

```
virtual void Update ( ) pure virtual
```

Update the draw.

◆ **IGaugeCCallback**

The **IGaugeCCallback** is an abstract base class that can be overridden. Implementors should override the function **CreateGaugeCDrawable(SINT32 id, const IGaugeCDrawableCreateParameters* pParameters)**.

See the Custom_Nav_Draw sample for an example use case.

Inherited by **ISerializableGaugeCCallback**.

Public Member Functions

```
virtual ULONG AddRef ()=0
virtual ULONG Release ()=0
virtual IGaugeCCallback * QueryInterface (LPCSTR pszInterface)=0
virtual void Update ()=0
virtual bool GetPropertyValue (SINT32 id, FLOAT64 *pValue)=0
virtual bool GetPropertyValue (SINT32 id, LPCSTR *pszValue)=0
virtual bool GetPropertyValue (SINT32 id, LPCWSTR *pszValue)=0
virtual bool SetPropertyValue (SINT32 id, FLOAT64 value)=0
virtual bool SetPropertyValue (SINT32 id, LPCSTR szValue)=0
virtual bool SetPropertyValue (SINT32 id, LPCWSTR szValue)=0
virtual IGaugeCDrawable * CreateGaugeCDrawable (SINT32 id, const
                                                IGaugeCDrawableCreateParameters *pParameters)=0
```

Member Function Documentation

◆ **AddRef()**

```
virtual ULONG AddRef ( ) pure virtual
```

Add a reference.

◆ **CreateGaugeCDrawable()**

```
virtual
IGaugeCDrawable*
CreateGaugeCDrawable ( SINT32           id,
                      const IGaugeCDrawableCreateParameters * pParameters
                    )

```

pure virtual

Create a gauge drawable.

Parameters

- id** The gauge drawable's id.
- pParameters** The parameters.

◆ **GetPropertyValues() [1 / 3]**

```
virtual bool GetPropertyValue ( SINT32    id,
                             FLOAT64   *pValue
                           )

```

pure virtual

Pass the FLOAT64 value by finding the corresponding id.

Parameters

id The property value's id.
pValue The property value's FLOAT64 value.

Returns

True if id and value found, false otherwise.

◆ **GetPropertyValue()** [2 / 3]

```
virtual bool GetPropertyValue ( SINT32 id,  
                           LPCSTR * pszValue  
                           )  
                           pure virtual
```

Pass the LPCSTR value by finding the corresponding id.

Parameters

id The property value's id.
pszValue The property value's LPCSTR value.

Returns

True if id and value found, false otherwise.

◆ **GetPropertyValue()** [3 / 3]

```
virtual bool GetPropertyValue ( SINT32 id,  
                           LPCWSTR * pszValue  
                           )  
                           pure virtual
```

Pass the LPCWSTR value by finding the corresponding id.

Parameters

id The property value's id.
pszValue The property value's LPCWSTR value.

Returns

True if id and value found, false otherwise.

◆ **QueryInterface()**

```
virtual IGaugeCCallback* QueryInterface ( LPCSTR pszInterface ) pure virtual
```

Query Interface.

◆ **Release()**

```
virtual ULONG Release ( ) pure virtual
```

Release the reference.

◆ **SetPropertyValue()** [1 / 3]

```
virtual bool SetPropertyValue ( SINT32 id,  
                           FLOAT64 value  
                           )  
                           pure virtual
```

Find the id and sets the FLOAT64 value that's passed in.

Parameters

id The property value's id.
value The property value's FLOAT64 value.

Returns

True if id found and value set, false otherwise.

◆ **SetPropertyValue()** [2 / 3]

```
virtual bool SetPropertyValue ( SINT32 id,  
                           LPCSTR szValue  
                           )  
                           pure virtual
```

Find the id and sets the LPCSTR value that's passed in.

Parameters

id The property value's id.
szValue The property value's LPCSTR value.

Returns

True if id found and value set, false otherwise.

◆ **SetPropertyValue()** [3 / 3]

```
virtual bool SetPropertyValue ( SINT32 id,  
                           LPCWSTR szValue  
                           )  
                           pure virtual
```

Find the id and sets the LPCWSTR value that's passed in.

Parameters

id The property value's id.
szValue The property value's LPCWSTR value.

Returns
True if id found and value set, false otherwise.

◆ **Update()**

virtual void Update() pure virtual

Update every 18hz.

◆ **ISerializableGaugeCCallback**

class ISerializableGaugeCCallback

Inherits [IGaugeCCallback](#).

Public Member Functions

virtual bool [Serialize](#)(NetOutPublic & netout)=0
 virtual bool [Deserialize](#)(NetInPublic & netin)=0

▶ [Public Member Functions inherited from IGaugeCCallback](#)

Member Function Documentation

◆ **Deserialize()**

virtual bool Deserialize(NetInPublic & [netin](#)) pure virtual

◆ **Serialize()**

virtual bool Serialize(NetOutPublic & [netout](#)) pure virtual

◆ **IAircraftCCallback**

class IAircraftCCallback

AircraftCallback is an abstract base class that can be overridden. Implementors should override the function [CreateGaugeCCallback\(\)](#).

See the Cabin_Conflict and Custom_Nav_Draw samples for example use cases.

Public Member Functions

virtual ULONG [AddRef](#)()=0
 virtual ULONG [Release](#)()=0
 virtual IAircraftCCallback * [QueryInterface](#)(LPCSTR pszInterface)=0
 virtual IGaugeCCallback * [CreateGaugeCCallback](#)()=0
 virtual void [Update](#)()=0

Member Function Documentation

◆ **AddRef()**

virtual ULONG AddRef() pure virtual

Add a reference.

◆ **CreateGaugeCCallback()**

virtual IGaugeCCallback* CreateGaugeCCallback() pure virtual

Create Gauge Callback.

◆ **QueryInterface()**

virtual IAircraftCCallback* QueryInterface(LPCSTR [pszInterface](#)) pure virtual

Query Interface.

Parameters
`pszInterface` The Interface.

◆ **Release()**

virtual ULONG Release() pure virtual

Release the reference.

◆ **Update()**

virtual void Update() pure virtual

◆ **IPanelCCallback**

class IPanelCCallback

PanelCallback is an abstract base class that can be overridden. Implementors should override the functions CreateAircraftCallback(UINT32 ContainerID) as well as GetPropertyTable.

See the Cabin_Comfort and Custom_Nav_Draw samples for example use cases.

Inherited by [IFSXPanelICCallback](#).

Public Member Functions

```
virtual ULONG AddRef ()=0  
virtual ULONG Release ()=0  
virtual IPanelCCallback * QueryInterface (LPCSTR pszInterface)=0  
virtual UINT32 GetVersion ()=0  
virtual IAircraftCCallback * CreateAircraftCCallback (UINT32 ContainerID)=0  
    virtual bool ConvertStringToProperty (LPCSTR keyword, SINT32 *pID)=0  
    virtual bool ConvertPropertyToString (SINT32 id, LPCSTR *pKeyword)=0  
    virtual bool GetPropertyUnits (SINT32 id, ENUM *pEnum)=0
```

Member Function Documentation

◆ AddRef()

```
virtual ULONG AddRef ( ) pure virtual
```

Add a reference.

◆ ConvertPropertyToString()

```
virtual bool ConvertPropertyToString ( SINT32 id,  
                                     LPCSTR * pKeyword  
                                     ) pure virtual
```

Convert property to string.

Parameters

id The id.
pKeyword The keyword.

Returns

True if id found and pKeyword value set, false otherwise.

◆ ConvertStringToProperty()

```
virtual bool ConvertStringToProperty ( LPCSTR keyword,  
                                      SINT32 * pID  
                                      ) pure virtual
```

Convert string to property.

Parameters

keyword The keyword.
pID The id.

Returns

True if keyword found and pID value set, false otherwise.

◆ CreateAircraftCCallback()

```
virtual IAircraftCCallback* CreateAircraftCCallback ( UINT32 ContainerID ) pure virtual
```

Create Aircraft Callback.

Parameters

ContainerID The containers id.

◆ GetPropertyUnits()

```
virtual bool GetPropertyUnits ( SINT32 id,  
                               ENUM * pEnum  
                               ) pure virtual
```

Get property units.

Parameters

id The id.
pEnum The enum.

Returns

True if id found and pEnum value set, false otherwise.

◆ GetVersion()

```
virtual UINT32 GetVersion ( ) pure virtual
```

Get Version.

◆ QueryInterface()

virtual IPanelCCallback* QueryInterface (LPCSTR pszInterface) pure virtual

Query Interface.

Parameters

pszInterface The Interface.

◆ Release()

virtual ULONG Release () pure virtual

Release the reference.

◆ IFSXPanelCCallback

class IFSXPanelCCallback

Inherits IPanelCCallback.

Public Member Functions

virtual void Clear ()=0

▶ Public Member Functions inherited from IPanelCCallback

Member Function Documentation

◆ Clear()

virtual void Clear () pure virtual

◆ P3D::IPanelSystemV400

class P3D::IPanelSystemV400

Parameter list interface used by IPanelSystemV400

Inherits IUnknown.

Private Member Functions

virtual BOOL IsPanelWindowVisibleIdent (UINT32 panel_id) PURE
virtual ENUM TooltipUnitsGetSet (int action, ENUM type) PURE
virtual void ElementListQuery (PELEMENT_HEADER element) PURE
virtual void ElementListInstall (PELEMENT_HEADER element, PVOID resource_file_handle) PURE
virtual void ElementListInitialize (PELEMENT_HEADER element) PURE
virtual void ElementListUpdate (PELEMENT_HEADER element) PURE
virtual void ElementListGenerate (PELEMENT_HEADER element, GENERATE_PHASE phase) PURE
virtual void ElementListPlot (PELEMENT_HEADER element) PURE
virtual void ElementListErase (PELEMENT_HEADER element) PURE
virtual void ElementListKill (PELEMENT_HEADER element) PURE
virtual void MouseListInstall (PMOUSERECT rect, PGAUGEHDR gauge_header, PPIXPOINT size) PURE
virtual void MouseListRegister (PMOUSERECT rect, PGAUGEHDR gauge_header) PURE
virtual void MouseListUnregister (PMOUSERECT rect, PGAUGEHDR gauge_header) PURE
virtual BOOL PanelWindowToggle (UINT32 panel_id) PURE
virtual ERR TriggerKeyEvent (ID32 event_id, UINT32 value) PURE
virtual void RegisterVarByName (PVOID var, VAR_TYPE var_type, LPSTR name) PURE
virtual void InitializeVar (PMODULE_VAR module_var) PURE
virtual void InitializeVarByName (PMODULE_VAR module_var, LPSTR name) PURE
virtual void LookupVar (PMODULE_VAR module_var) PURE
virtual void UnregisterVarByName (LPSTR name) PURE
virtual void UnregisterAllNamedVars (void) PURE
virtual BOOL PanelWindowCloseIdent (UINT32 panel_id) PURE
virtual BOOL PanelWindowOpenIdent (UINT32 panel_id) PURE
virtual void PanelWindowToggleHudColor (void) PURE
virtual void PanelWindowToggleHudUnits (void) PURE
virtual void RadioStackPopup (void) PURE
virtual void RadioStackAutoclose (void) PURE
virtual ID CheckNamedVariable (LPCSTR name) PURE
virtual ID RegisterNamedVariable (LPCSTR name) PURE
virtual FLOAT64 GetNamedVariableValue (ID id) PURE
virtual FLOAT64 GetNamedVariableTypedValue (ID id, ENUM units) PURE
virtual void SetNamedVariableValue (ID id, FLOAT64 value) PURE
virtual void SetNamedVariableTypedValue (ID id, FLOAT64 value, ENUM units) PURE
virtual LPCSTR GetNameOfNamedVariable (ID id) PURE
virtual LPCTSTR PanelResourceStringGet (ID32 id) PURE
virtual BOOL PanelWindowToggleMenuId (ID32 menu_id) PURE
virtual void ElementUseColor (PELEMENT_HEADER element, BOOL override, UINT32 color) PURE
virtual void SetGaugeFlags (LPCSTR name, FLAGS32 newflags) PURE
virtual FLAGS32 GetGaugeFlags (LPCSTR name) PURE
virtual BOOL GaugeCalculatorCodePrecompile (LPCSTR *pCompiled, UINT32 *pCompiledSize, LPCSTR source) PURE
virtual BOOL ExecuteCalculatorCode (LPCSTR code, FLOAT64 *value, SINT32 *value, LPCSTR *value) PURE
virtual BOOL FormatCalculatorString (LPSTR result, UINT32 resultsize, LPCSTR format) PURE

```

virtual ENUM GetUnitsEnum (LPCSTR unitname) PURE
virtual ENUM GetAircraftVarEnum (LPCSTR simvar) PURE
virtual FLOAT64 AircraftVarget (ENUM simvar, ENUM units, SINT32 index) PURE
virtual BOOL PanelRegisterCCallback (LPCSTR name, IPanelCCallback *pcallback) PURE
virtual IPanelCCallback * PanelGetRegisteredCCallback (LPCSTR name) PURE
virtual IAircraftCCallback * PanelGetAircraftCCallback (LPCSTR name) PURE
virtual void SendKeyEvent (ID32 event_id, UINT32 value) PURE
virtual void RegisterKeyEventHandler (GAUGE_KEY_EVENT_HANDLER handler, PVOID userdata) PURE
virtual void UnregisterKeyEventHandler (GAUGE_KEY_EVENT_HANDLER handler, PVOID userdata) PURE
virtual BOOL ProcessSharedEventOut (PGaugeHDR gauge_header, BYTE *pBuf, UINT32 nSize) PURE
virtual BOOL IsMaster () PURE
virtual void SetNamedVariableValueSync (ID id, FLOAT64 value) PURE
virtual void SetNamedVariableSyncEnabled (ID id, BOOL bSync) PURE
virtual LPCTSTR GetEventDescription (ID32 event_id) PURE
virtual LPCTSTR GetEventTokenString (ID32 event_id) PURE
virtual HRESULT QueryPdk (REFIID iid, void **ppPdk) PURE
virtual UINT32 GetEventCount () PURE
virtual ID32 GetEventIdByIndex (UINT32 ulIndex) PURE

```

Member Function Documentation

◆ AircraftVarget()

```

virtual FLOAT64 AircraftVarget ( ENUM simvar,
                                 ENUM units,
                                 SINT32 index
                               ) [private] [virtual]

```

Retrieves the value of an aircraft simulation variable.

Parameters

[in] **simvar** Specifies a simulation variable enum value. Use get_aircraft_var_enum to retrieve the enum value from a string.
 [in] **units** Specifies the units enum value the returned value should have. Use get_units_enum to retrieve the enum value from a string.
 [in] **index** Specifies an index number, which is required for some engine and communication data. Refer to the Simulation Variables document for details.

Returns

The function returns the value in a FLOAT64. If the simulation variable is not found, zero will be returned.

◆ CheckNamedVariable()

```

virtual ID CheckNamedVariable ( LPCSTR name ) [private] [virtual]

```

Retrieves the ID number of a named local variable, if it exists.

Parameters

[in] **name** Specifies the variable name.

Returns

The function returns an ID number if the variables exists, or -1 if it does not.

Remarks

Local variable names are case-insensitive.

◆ ElementListErase()

```

virtual void ElementListErase ( PELEMENT_HEADER element ) [private] [virtual]

```

Function used before redrawing a gauge.

Parameters

[in] **element** A pointer to the element header.

Remarks

It is not necessary to call this function if the redrawing of the gauge is handled by ElementListPlot.

◆ ElementListGenerate()

```

virtual void ElementListGenerate ( PELEMENT_HEADER element,
                                   GENERATE_PHASE phase
                                 ) [private] [virtual]

```

Function to regenerate the effects mask and images for a gauge.

Parameters

[in] **element** A pointer to the element header.
 [in] **phase** Unused, enter zero.

Remarks

See the remarks for ElementListPlot.

◆ ElementListInitialize()

```

virtual void ElementListInitialize ( PELEMENT_HEADER element ) [private] [virtual]

```

Function to prepare a gauge before being redrawn.

Parameters

[in] **element** A pointer to the element header.

Remarks

See the remarks for ElementListPlot.

◆ ElementListInstall()

```
virtual void ElementListInstall( PELEMENT_HEADER element,
                                PVOID             resource_file_handle
                               )
```

private **virtual**

Function used to install a gauge during initialization of a panel.

Parameters

[in] **element** A pointer to the element header.
 [in] **resource_file_handle** Specifies the resource file handle.

Remarks

Use this function before calling any of the other element_list functions. A gauge is defined by an element list, however the macros used to create gauges described in the tutorial mean that in many cases these functions do not need to be called directly.

◆ ElementListKill()

```
virtual void ElementListKill( PELEMENT_HEADER element )
```

private **virtual**

Function used to remove a gauge completely, for example when changing user aircraft.

Parameters

[in] **element** A pointer to the element header.

◆ ElementListPlot()

```
virtual void ElementListPlot( PELEMENT_HEADER element )
```

private **virtual**

Function to redraw a gauge.

Parameters

[in] **element** A pointer to the element header.

Remarks

Before calling this function, each and every time, call the following functions in this order:

1. ElementListInitialize
2. ElementListUpdate
3. ElementListGenerate
4. ElementListPlot

◆ ElementListQuery()

```
virtual void ElementListQuery( PELEMENT_HEADER element )
```

private **virtual**

Function used to register the failure conditions for a gauge.

Parameters

[in] **element** A pointer to the element header.

Remarks

Call this function once, after ElementListInstall.

◆ ElementListUpdate()

```
virtual void ElementListUpdate( PELEMENT_HEADER element )
```

private **virtual**

Function to calculate new values for each gauge element.

Parameters

[in] **element** A pointer to the element header.

Remarks

See the remarks for ElementListPlot.

◆ ElementUseColor()

```
virtual void ElementUseColor( PELEMENT_HEADER element,
                            BOOL            override,
                            UINT32          color
                           )
```

private **virtual**

Selects the text color for string elements.

Parameters

[in] **element** A pointer to the element header.
 [in] **override** Specifies if a color selected by the user should be overridden.
 [in] **color** RGB value for the color.

◆ ExecuteCalculatorCode()

```
virtual BOOL ExecuteCalculatorCode( LPCSTR   code,
                                  FLOAT64 * fvalue,
                                  SINT32 *  ivalue,
                                  LPCSTR *  svalue
                                 )
```

private **virtual**

Evaluates a coded calculator string.

Parameters

[in] **code** Specifies the calculator code.
[out] **fvalue** Pointer to a float. Returns the result of the calculation, if it is a floating point value.
[out] **ivalue** Pointer to an integer. Returns the result of the calculation, if it is an integer value.
[out] **svalue** Pointer to a string. Returns the result of the calculation, if it is a string.

Returns

If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

Remarks

Example:

```
FLOAT64 att_pitch = 0;  
FLOAT64 att_bank = 0;  
ExecuteCalculatorCode ("(A:ATTITUDE INDICATOR PITCH DEGREES:2, degrees)", &att_pitch, NULL, NULL);  
ExecuteCalculatorCode ("(A:ATTITUDE INDICATOR BANK DEGREES:2, degrees)", &att_bank, NULL, NULL);
```

◆ FormatCalculatorString()

```
virtual BOOL FormatCalculatorString (LPSTR result,  
                                    UINT32 resultsize,  
                                    LPCSTR format  
                                )
```

private **virtual**

Evaluates a formatted calculator string.

Parameters

[out] **result** Returns the formatted string.
[out] **resultsize** Returns the length of the formatted string.
[in] **format** Specifies the calculator string to format.

Returns

If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

◆ GaugeCalculatorCodePrecompile()

```
virtual BOOL GaugeCalculatorCodePrecompile (LPCSTR * pCompiled,  
                                         UINT32 * pCompiledSize,  
                                         LPCSTR source  
                                       )
```

private **virtual**

Compresses a calculator string into a more efficient internal format.

Parameters

[in] **pCompiled** Pointer to a string, which will contain the compiled string if the function call is successful.
[out] **pCompiledSize** Pointer to an integer, which will contain the length of the compiled string if the function call is successful.
[out] **source** Specifies the source calculator string. Refer to the Creating XML Gauges document for details on format strings.

Returns

If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

◆ GetAircraftVarEnum()

```
virtual ENUM GetAircraftVarEnum (LPCSTR simvar)
```

private **virtual**

Retrieves the enum value for a simulation variable, given the string name of that variable.

Parameters

[in] **simvar** Specifies the simulation variable name.

Returns

The function returns the ENUM value of the simulation variable, or SIMVAR_NONE (-1) if the variable name is not found.

Remarks

Use this function before calling AircraftVarget.

◆ GetEventCount()

```
virtual UINT32 GetEventCount ( )
```

private **virtual**

Returns the number of event tokens. The event count can be used to iteratively call [GetEventIdByIndex\(\)](#) to retrieve event id's.

Returns

The number of event tokens.

◆ GetEventDescription()

```
virtual LPCTSTR GetEventDescription (ID32 event_id)
```

private **virtual**

Returns the description for the given key event id as seen in the Control Settings user interface.

Parameters

[in] **event_id** The id of the key event.

Returns

The function returns the description of the given key event id. The function returns NULL if the description or key event id is not found.

◆ GetEventIdByIndex()

```
virtual ID32 GetEventIdByIndex (UINT32 uIndex)
```

private **virtual**

Returns the event id for the associated event index. The event count can be obtained from the [GetEventCount\(\)](#) function.

Parameters

[in] **uIndex** The index of the event id being requested.

Returns
The event id for the associated index. Valid event id's are greater than zero.

◆ **GetEventTokenString()**

virtual LPCTSTR GetEventTokenString (ID32 event_id) [private] [virtual]

Returns the token string for the given key event id.

Parameters
[in] **event_id** The id of the key event.

Returns
The function returns the token string of the given key event id. The function returns NULL if the key event id is not found.

◆ **GetGaugeFlags()**

virtual FLAGS32 GetGaugeFlags (LPCSTR name) [private] [virtual]

Used to retrieve the flags set on a gauge.

Parameters
[in] **name** Specifies the name of the gauge.

Returns
The function returns a FLAGS32 value containing the flags that are set.

Remarks
Flag Values:

GAUGE_FLAG_NORMAL	0
GAUGE_FLAG_HIDDEN	0x1
GAUGE_FLAG_BLINKING	0x2
GAUGE_FLAG_GRAYED	0x4
GAUGE_FLAG_HIGHLIGHTED	0x8

◆ **GetNamedVariableTypedValue()**

virtual FLOAT64 GetNamedVariableTypedValue (ID id, ENUM units) [private] [virtual]

Retrieves the value of a named local variable, in the specified units.

Parameters
[in] **id** Specifies the ID of the variable.
[in] **units** Specifies the enum value of the units required. Use get_units_enum to retrieve the enum value from a string.

Returns
The function returns the value in a FLOAT64. Zero is returned if the variable ID is not found.

◆ **GetNamedVariableValue()**

virtual FLOAT64 GetNamedVariableValue (ID id) [private] [virtual]

Retrieves the value of a local variable, given an ID.

Parameters
[in] **id** Specifies the ID of the variable.

Returns
The function returns the value in a FLOAT64. Zero is returned if the variable ID is not found.

◆ **GetNameOfNamedVariable()**

virtual LPCSTR GetNameOfNamedVariable (ID id) [private] [virtual]

Retrieves the name of a local variable, given an ID number.

Parameters
[in] **id** Specifies the ID of the variable.

Returns
The function returns the name in a PCSTRINGZ, or NULL if the name is not found.

◆ **GetUnitsEnum()**

virtual ENUM GetUnitsEnum (LPCSTR unitname) [private] [virtual]

Retrieves the enum value for units, given the string name of the units.

Parameters
[in] **unitname** Specifies the string name of the units.

Returns
The function returns the ENUM. value for the units, or UNITS_UNKNOWN (-1) if the string name is not found.

Remarks
Use this function before calling AircraftVarGet.

◆ **InitializeVar()**

virtual void InitializeVar (PMODULE_VAR module_var) [private] [virtual]

Initializes a token variable.

Parameters

[in] **module_var** Specifies a pointer to a **MODULE_VAR** structure, containing the token variable to initialize.

Remarks

Before a token variable can be used, you must initialize it with a call to this function.

◆ **InitializeVarByName()**

```
virtual void InitializeVarByName( PMODULE_VAR module_var,  
                                LPSTR name  
                                )  
                                [private] [virtual]
```

Initializes a **MODULE_VAR** structure, given the name of a token variable.

Parameters

[in] **module_var** Specifies the address of the **MODULE_VAR** structure that will receive information about the variable.
[in] **name** Specifies the name of the variable (the same name used in register_var_by_name).

◆ **IsMaster()**

```
virtual BOOL IsMaster( ) [private] [virtual]
```

Returns true if the aircraft is the master aircraft of a shared cockpit.

Returns

If the aircraft is the master aircraft, it returns a non-zero value. If it is not, it returns zero.

Remarks

This function is used in the multiplayer scenario of a shared cockpit. One aircraft is the master and one is not. There is a maximum of two users in this scenario.

◆ **IsPanelWindowVisibleIdent()**

```
virtual BOOL IsPanelWindowVisibleIdent( UINT32 panel_id ) [private] [virtual]
```

Returns if the specified panel window is visible.

Parameters

panel_id Specifies the identification number of the window to query. The identification number is specified in the Panel.cfg file in the [WindowXX] section by the variable ident.

Returns

If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

◆ **LookupVar()**

```
virtual void LookupVar( PMODULE_VAR module_var ) [private] [virtual]
```

Updates the contents of a token variable.

Parameters

[in] **module_var** Pointer to a **MODULE_VAR** structure, containing the token variable to update.

Returns

Before using the contents of a module variable, you must call the **LookupVar** function.

◆ **MouseListInstall()**

```
virtual void MouseListInstall( PMOUSERECT rect,  
                             PGAUGEHDR gauge_header,  
                             PPIXPOINT size  
                             )  
                             [private] [virtual]
```

Creates the mouse rectangles for a gauge.

Parameters

[in,out] **rect** Specifies a pointer to a list of **MOUSERECT** structures. The first rectangle in the list is the main box. The rectangles are scaled according to the settings in the gauge_header. The last rectangle structure in the list should have the type **MOUSE_RECT_EOL**.

[in] **gauge_header** Specifies a pointer to a gauge_header structure (defined in **gauges.h**).

[in] **size** Specifies a pointer to a **PIXPOINT** structure, which contains an x and y value, defining the size of the rectangle.

Remarks

Call **MouseListRegister** after setting up the mouse rectangles with this function. Note that the mouse rectangle creation macros can be used instead of these low level function calls.

◆ **MouseListRegister()**

```
virtual void MouseListRegister( PMOUSERECT rect,  
                               PGAUGEHDR gauge_header  
                               )  
                               [private] [virtual]
```

Registers the mouse windows.

Parameters

[in] **rect** Specifies a pointer to a **MOUSERECT** structure.

[in] **gauge_header** Specifies a pointer to a gauge_header structure (defined in **gauges.h**).

Returns◆ **MouseListUnregister()**

```
virtual void MouseListUnregister( PMOUSERECT rect,
```

P_GAUGEHDR *gauge_header*

)

private **virtual**

Unregisters the mouse windows.

Parameters

[**in**] **rect** Specifies a pointer to a **MOUSERECT** structure.
[**in**] **gauge_header** Specifies a pointer to a **gauge_header** structure (defined in **gauges.h**).

Returns

◆ **PanelGetAircraftCCallback()**

virtual **I****AircraftCCallback*** **PanelGetAircraftCCallback** (**LPCSTR** *name*) **private** **virtual**

Retrieves a pointer to the aircraft callback function.

Parameters

[**in**] **name** Specifies the name of the callback function.

Returns

The function returns a pointer to the **I****AircraftCCallback** function, or NULL if the name is not found.

◆ **PanelGetRegisteredCCallback()**

virtual **IPanelICCallback*** **PanelGetRegisteredCCallback** (**LPCSTR** *name*) **private** **virtual**

Retrieves a pointer to the registered callback function.

Parameters

[**in**] **name** Specifies the name of the module, "CABIN" in the Cabin_Comfort.cpp example.

Returns

The function returns a pointer to an **IPanelICCallback** function.

◆ **PanelRegisterCCallback()**

virtual **BOOL** **PanelRegisterCCallback** (**LPCSTR** *name*,
IPanelICCallback* *pcallback*) **private** **virtual**

Specifies the registered callback function.

Parameters

[**in**] **name** Specifies the name of the module, "CABIN" in the Cabin_Comfort.cpp example.
[**in**] **pcallback** Specifies a pointer to the **IPanelICCallback** function.

Returns

If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

◆ **PanelResourceStringGet()**

virtual **LPCTSTR** **PanelResourceStringGet** (**ID32** *id*) **private** **virtual**

Retrieves the resource string, given an ID.

Parameters

[**in**] **id** Specifies the resource ID.

Returns

The function returns the resource string in a **PCSTRINGZ**, or NULL if the ID is not found.

◆ **PanelWindowCloseIdent()**

virtual **BOOL** **PanelWindowCloseIdent** (**UINT32** *panel_id*) **private** **virtual**

Closes the specified panel window.

Parameters

[**in**] **panel_id** Specifies the identification number of the window to close. The identification number is specified in the Panel.cfg file in the [WindowXX] section by the variable ident.

Returns

If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

◆ **PanelWindowOpenIdent()**

virtual **BOOL** **PanelWindowOpenIdent** (**UINT32** *panel_id*) **private** **virtual**

Displays the specified panel window.

Parameters

[**in**] **panel_id** Specifies the identification number of the window to open. The identification number is specified in the Panel.cfg file in the [WindowXX] section by the variable ident.

Returns

If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

◆ **PanelWindowToggle()**

virtual **BOOL** **PanelWindowToggle** (**UINT32** *panel_id*) **private** **virtual**

Toggles the visible state of a panel window.

Parameters
[in] **panel_id** Specifies the identification number of the window to toggle. The identification number is specified in the Panel.cfg file in the [WindowXX] section by the variable ident.

Returns
If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

◆ **PanelWindowToggleHudColor()**

virtual void PanelWindowToggleHudColor (void) [private] [virtual]

Changes the global HUD color to the next in the list.

Parameters
none

Remarks
This function cycles through the range of HUD colors (green, dark green, blue, dark blue, red, dark red, black, white), setting the global HUD color to the next in the list..

◆ **PanelWindowToggleHudUnits()**

virtual void PanelWindowToggleHudUnits (void) [private] [virtual]

Toggles the HUD units between metric and imperial.

Parameters
none

◆ **PanelWindowToggleMenuId()**

virtual **BOOL** PanelWindowToggleMenuId (**ID32** menu_id) [private] [virtual]

Selects a menu item given an ID.

Parameters
[in] **menu_id** Specifies the menu ID

Returns
If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

Remarks
This function does not "toggle", but rather "selects", but the name is left unchanged for backwards compatibility.

◆ **ProcessSharedEventOut()**

virtual **BOOL** ProcessSharedEventOut (**PGAUGEHDR** gauge_header,
 BYTE * pBuf,
 UINT32 nSize
) [private] [virtual]

Sends data to the other aircraft, in a multiplayer shared cockpit scenario.

Parameters
[in] **gauge_header** Specifies the gauge header, which is used to identify the gauge that the event applies to.
[in] **pBuf** A pointer to an array of data.
[in] **nSize** The length of the data array, in bytes.

Returns
If the function succeeds, it returns a non-zero value. If it fails, it returns zero.

Remarks
This function is used to send data from one aircraft to another in the shared cockpit scenario of multiplayer operations. Only two aircraft can share a cockpit, one is the master and the other is not. This function can be used to send data from either one to the other.

◆ **QueryPdk()**

virtual **HRESULT** QueryPdk (**REFIID** iid,
 void ** ppPdk
) [private] [virtual]

Gets an **IPdk** pointer which can be used to query additional SDK services.

Parameters
[in] **iid** Specifies **IPdk** interface ID requested. IID_IPdkV01 is the latest version.
[out] **ppPdk** Specifies the address of the **IPdk** pointer to be set.

Returns
S_OK (0) if no errors and a valid **IPdk** pointer is set.

◆ **RadioStackAutoclose()**

virtual void RadioStackAutoclose (void) [private] [virtual]

Closes the radio stack if it was opened with RadioStackPopup.

Parameters
none

◆ **RadioStackPopup()**

virtual void RadioStackPopup (void) [private] [virtual]

Displays the radio stack.

Parameters**none****Remarks**

This function has no effect if the radio stack is already displayed.

◆ RegisterKeyEventHandler()

```
virtual void RegisterKeyEventHandler ( GAUGE_KEY_EVENT_HANDLER handler,
                                     PVOID                 userdata
                                     )
```

private **virtual**

Registers a key event callback function.

Parameters

[in] **handler** Specifies the handler function, which should match the following definition:

```
typedef void (*GAUGE_KEY_EVENT_HANDLER) (ID32 event, UINT32 evdata, PVOID userdata);
```

[in] **userdata** Specifies an optional value for use by the gauge developer. This value will be returned to the key event handler function, whenever it is called.

◆ RegisterNamedVariable()

```
virtual ID RegisterNamedVariable (LPCSTR name) private virtual
```

Registers a local variable name.

Parameters

[in] **name** Specifies the variable name.

Returns

The function returns an ID. If the named variable already exists, its existing ID will be returned. If it does not exist, a new registered variable is created.

Remarks

Local variable names are case-insensitive. The value of the variable is set to zero, and the units to UNITS_UNKNOWN, on creation.

◆ RegisterVarByName()

```
virtual void RegisterVarByName ( PVOID      var,
                               VAR_TYPE   var_type,
                               LPSTR      name
                               )
```

private **virtual**

Registers a variable from another gauge, for use by this gauge.

Parameters

[in] **var** Specifies the address of the variable.

[in] **var_type** Specifies the type of the variable, one of the following enum:

```
typedef enum VAR_TYPE {
    VAR_TYPE_NONE,
    TYPE_BOOL8,
    TYPE_UINT8,
    TYPE_SINT8,
    TYPE_FLAGS8,
    TYPE_ENUM8,
    TYPE_BOOL16,
    TYPE_ANGLE16,
    TYPE_UINT16,
    TYPE_SINT16,
    TYPE_UIF16,
    TYPE_SIF16,
    TYPE_FLAGS16,
    TYPE_ENUM16,
    TYPE_BCD16,
    TYPE_BCO16,
    TYPE_VAR16,
    TYPE_BOOL32,
    TYPE_ANGLE32,
    TYPE_UINT32,
    TYPE_SINT32,
    TYPE_UIF32,
    TYPE_SIF32,
    TYPE_FLAGS32,
    TYPE_ENUM32,
    TYPE_VAR32,
    TYPE_ANGLE48,
    TYPE_SINT48,
    TYPE_UIF48,
    TYPE_SIF48,
    TYPE_UINT64,
    TYPE_SINT64,
    TYPE_SIF64,
    TYPE_FLOAT64,
    TYPE_BOOL,
    TYPE_FLAGS,
    TYPE_ENUM,
    TYPE_VOID,
    TYPE_PVOID,
    TYPE_PUINT32,
    TYPE_PSINT32,
    TYPE_PFLOAT64,
    VAR_TYPE_MAX
} VAR_TYPE;
```

Remarks

You can use named variables to enable communication between two or more gauges. To establish communication between gauges, a "server" and "client" gauge needs to be defined. The terms "server" and "client" just distinguish between variable ownership and variable usage:

*The server gauges provides one or more named variables for other gauges to access.

*The client gauges accesses one or more named variables from the server gauges.

A single gauge can be both a server (by providing one or more variables) and a client (by accessing another gauge's variables) at the same time. Use the

RegisterVarByName, UnregisterVarByName, and InitializeVarByName functions with named variables. The server gauge uses the RegisterVarByName function to register a named variable with the panel system at startup, so create a callback for your gauge as part of the gauge_header structure. You can set this so it performs at startup, on shutdown, etc. When using named variables, don't call the LookupVar function (as you would with the standard panel system variables). After InitializeVarByName is called, the var_ptr field of the MODULE_VAR structure contains a pointer to the named variable. The panel system doesn't recognize named variables, per se, but the system does maintain the name to the pointer link for gauges to query. As a result, you can't use a named variable as a controlling variable for an ELEMENT structure directly. Instead, use a MODULE_VAR_NONE structure and provide a callback function that can query the variable's value using the var_ptr field of the MODULE_VAR structure. Because named variables work via direct pointers between gauges, make sure that the server gauge is loaded before, or at the same time as, the client gauge. You can make sure this happens by either putting both gauges on the same panel window or by putting the server gauge on the main panel window. This ensures that the server gauge is loaded and the named variable is registered before the client gauge tries to connect to it. Alternatively, you can check the returned var_ptr for NULL and the returned var_type (both in the MODULE_VAR structure) for VAR_TYPE_NONE and execute (in the ELEMENT callback function) the InitializeVarByName function until it returns valid information. (You can also call the InitializeVarByName function every time you want to access the variable, but this approach is a little slower than caching the information once it's received.). The server gauge must keep checking the current value of the variable(s) it has made available, if the current state/value of that variable has any effect. You can use named variables at any point in the simulation when you want to pass information between two or more gauges. Because named variables are shared by direct pointer access, you can also share out an entire data structure using one named variable, as long as the server and client gauges interpret the same data format. You can place these gauges anywhere on a panel, as long as the server gauge is guaranteed to load before or at the same time as the client gauge.

◆ SendKeyEvent()

```
virtual void SendKeyEvent( ID32 event_id,
                           UINT32 value
                         )
```

private **virtual**

Transmits a WM_COMMAND application event.

Parameters

[in] **event_id** Specifies a WM_COMMAND event ID.
 [in] **value** Specifies a value to be transmitted along with the event ID. Can be set to zero.

◆ SetGaugeFlags()

```
virtual void SetGaugeFlags( LPCSTR name,
                           FLAGS32 newflags
                         )
```

private **virtual**

Used to specify the flags for a gauge.

Parameters

[in] **name** Specifies the name of the gauge.
 [in] **newflags** One or more of the following flags:

Remarks

Flag Values:

GAUGE_FLAG_NORMAL	0
GAUGE_FLAG_HIDDEN	0x1
GAUGE_FLAG_BLINKING	0x2
GAUGE_FLAG_GRAYED	0x4
GAUGE_FLAG_HILIGHTED	0x8

◆ SetNamedVariableSyncEnabled()

```
virtual void SetNamedVariableSyncEnabled( ID id,
                                         BOOL bSync
                                       )
```

private **virtual**

Enables a specified named variable to sync from a shared cockpit.

Parameters

[in] **id** Specifies the ID of the variable.
 [in] **bSync** Sets whether to sync variable or not.

◆ SetNamedVariableTypedValue()

```
virtual void SetNamedVariableTypedValue( ID id,
                                         FLOAT64 value,
                                         ENUM units
                                       )
```

private **virtual**

Specifies a local variable should be set to the given value with the given units.

Parameters

[in] **id** Specifies the ID of the variable.
 [in] **value** Specifies the value the variable should be set to.
 [in] **units** Specifies the units of the value.

◆ SetNamedVariableValue()

```
virtual void SetNamedVariableValue( ID id,
                                   FLOAT64 value
                                 )
```

private **virtual**

Sets a local variable to a given value.

Parameters

[in] **id** Specifies the ID of the variable.
 [in] **value** Specifies the value the variable should be set to.

Returns

◆ SetNamedVariableValueSync()

```
virtual void SetNamedVariableValueSync( ID id,
```

FLOAT64 value
)

Syncs a specified named variable from a shared cockpit.

Parameters

- [in] **id** Specifies the ID of the variable.
- [in] **value** Value of variable to sync.

◆ TooltipUnitsGetSet()

```
virtual ENUM TooltipUnitsGetSet( int     action,
                                ENUM type
)                                     private virtual
```

Specifies or retrieves the tooltip units (metric or US).

Parameters

- [in] **action** Specifies the action. If this value is less than zero, the units are toggled (between metric and US imperial). If this value equals zero, the enum value for the units is returned. If this value is greater than zero, the units are set to the value of the type parameter.
- [in] **type** The enum value to set the tooltip units to, one of the below TOOLTIP_UNITS_TYPE enumeration types.

Returns

The function returns one member of the TOOLTIP_UNITS_TYPE enumeration below.

Remarks

```
enum TOOLTIP_UNITS_TYPE
{
    TOOLTIP_UNITS_TYPE_DEFAULT,
    TOOLTIP_UNITS_TYPE_METRIC,
    TOOLTIP_UNITS_TYPE_US,
}
```

◆ TriggerKeyEvent()

```
virtual ERR TriggerKeyEvent( ID32 event_id,
                            UINT32 value
)                                     private virtual
```

Initiates the action of a key event.

Parameters

- [in] **event_id** Specifies the event ID. Refer to the list of key events in the EventIDs document, and the #define KEY_events in **gauges.h**.
- [in] **value** Specifies an additional integer value. Set this to zero if it is not required.

Returns

The function returns an ERR, which is usually ignored. If the event requested is not appropriate, it simply will not happen.

◆ UnregisterAllNamedVars()

```
virtual void UnregisterAllNamedVars( void ) private virtual
```

Unregisters all token variables, and frees up the memory used.

Parameters

- none**

◆ UnregisterKeyEventHandler()

```
virtual void UnregisterKeyEventHandler( GAUGE_KEY_EVENT_HANDLER handler,
                                         PVOID             userdata
)                                     private virtual
```

Unregisters the key event handler.

Parameters

- [in] **handler** Specifies the handler function.
- [in] **userdata** Specifies the user data value specified when creating the handler function.

◆ UnregisterVarByName()

```
virtual void UnregisterVarByName( LPSTR name ) private virtual
```

Unregisters a named variable from another gauge, and frees up the memory used.

Parameters

- [in] **name** Specifies the name of the variable.

Macros

```
#define ISERIALIZABLE_GAUGECALLBACK_NAME "ISerializableGaugeCCallback"
#define IFSX_PANELCCALLBACK_NAME "IFSXPanelCCallback"
#define DECLARE_PANEL_CALLBACK_REFCOUNT(CLASSNAME)
#define DEFINE_PANEL_CALLBACK_REFCOUNT(CLASSNAME)
#define INIT_PANEL_CALLBACK_REFCOUNT(CLASSNAME) m_RefCount = 1;
```

Variables

GUID **IID_IPanelSystemV400**
 Current version of PanelSystem interface. More...

GUID **SID_PanelSystem**
 PanelSystem service ID. More...

Macro Definition Documentation

◆ DECLARE_PANEL_CALLBACK_REFCOUNT

```
#define DECLARE_PANEL_CALLBACK_REFCOUNT( CLASSNAME )
```

Value:

```
private:           \
    ULONG   m_RefCount;      \
public:          \
    ULONG AddRef ();        \
    ULONG Release ();       \
```

◆ DEFINE_PANEL_CALLBACK_REFCOUNT

```
#define DEFINE_PANEL_CALLBACK_REFCOUNT( CLASSNAME )
```

Value:

```
ULONG CLASSNAME::AddRef ()           \
{                                     \
    return ++m_RefCount;            \
}                                     \
ULONG CLASSNAME::Release ()          \
{                                     \
    ULONG result = --m_RefCount;    \
    if (result < 1)                \
        delete this;               \
    return result;                 \
```

◆ IFSX_PANELCCALLBACK_NAME

```
#define IFSX_PANELCCALLBACK_NAME "IFSXPanelCCallback"
```

◆ INIT_PANEL_CALLBACK_REFCOUNT

```
#define INIT_PANEL_CALLBACK_REFCOUNT( CLASSNAME ) m_RefCount = 1;
```

◆ ISERIALIZABLE_GAUGECCALLBACK_NAME

```
#define ISERIALIZABLE_GAUGECCALLBACK_NAME "ISerializableGaugeCCallback"
```

Variable Documentation

◆ IID_IPanelSystemV400

```
GUID IID_IPanelSystemV400
```

Current version of PanelSystem interface.

◆ SID_PanelSystem

```
GUID SID_PanelSystem
```

PanelSystem service ID.

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Environment Force Service

Overview

This PDK service allows callers to manipulate environment forces during runtime. Environment forces registered and unregistered using this interface. Environment forces utilize world-based positioning and may or may not be associated with an object.

Classes

- class [IEnvironmentForceCallbackV410](#)
- class [IEnvironmentForceManagerV410](#)
- class [IEnvironmentForceV400](#)

Class Documentation

◆ [P3D::IEnvironmentForceCallbackV410](#)

class P3D::IEnvironmentForceCallbackV410

The interface used when registering for environment force notifications.

Inherits [IUnknown](#).

Private Member Functions

```
virtual void Invoke (_in UINT uObjectId, _in const IEnvironmentForceV400 *const *ppEnvironmentForces, _in UINT uCount) override
```

Member Function Documentation

◆ [Invoke\(\)](#)

```
virtual  
void  
Invoke (_in UINT uObjectId,  
         _in const IEnvironmentForceV400 *const * ppEnvironmentForces,  
         _in UINT uCount)  
)
```

[private](#) [pure virtual](#)

This function is called when one or more environment forces are within the given radius of the given object.

Parameters

uObjectId	The object id that was registered to receive the callback.
ppEnvironmentForces	The array of environment forces within the registered radius.
uCount	The number of environment forces in the array.

Remarks

The returned environment force vector includes forces associated with the registered object. These forces can be filtered using the [IEnvironmentForceV400::GetOwnerId\(\)](#) function.

◆ [P3D::IEnvironmentForceManagerV410](#)

class P3D::IEnvironmentForceManagerV410

This service is responsible for the management of environment forces.

Inherits [IUnknown](#).

Private Member Functions

```
virtual HRESULT RegisterEnvironmentForce (_in __notnull IEnvironmentForceV400 *pEnvironmentForce)  
PURE  
virtual HRESULT UnregisterEnvironmentForce (_in __notnull IEnvironmentForceV400  
*pEnvironmentForce) PURE  
virtual HRESULT GetForcesInRadius (_in const DXYZ &vWorldPosRadiansFeet, _in float fRadiusFeet, _in  
IEnvironmentForceList &vecForces) const PURE  
virtual HRESULT RegisterNotification (_in UINT uObjectId, _in float fRadiusFeet, _in __notnull  
IEnvironmentForceCallbackV410 *nfCallback) PURE
```

virtual HRESULT UnregisterNotification (__in UINT uObjectId, __in __notnull IEnvironmentForceCallbackV410 *pfCallback) PURE

Member Function Documentation

◆ GetForcesInRadius()

virtual HRESULT
GetForcesInRadius (__in const DXYZ & vWorldPosRadiansFeet,
__in float fRadiusFeet,
__in __notnull IEnvironmentForceList & vecForces
)

private virtual

Returns a list of forces for a given radius.

Parameters

vWorldPosRadiansFeet The world location in radians/feet.
fRadiusFeet The radius in feet.
vecForces The list of forces.

Returns

This function currently only returns S_OK.

◆ RegisterEnvironmentForce()

virtual HRESULT
RegisterEnvironmentForce (__in __notnull IEnvironmentForceV400 * pEnvironmentForce) private virtual

Registers a developer defined environment force.

Parameters

pEnvironmentForce The IEnvironmentForceV400 to be registered.

Returns

S_OK if the environment force was successfully registered, E_FAIL otherwise.

◆ RegisterNotification()

virtual HRESULT
RegisterNotification (__in UINT uObjectId,
__in float fRadiusFeet,
__in __notnull IEnvironmentForceCallbackV410 * pfCallback
)

private virtual

Registers an environment force notification callback. The callback is issued when one or more environment forces are within the given radius of the given object.

Parameters

uObjectId The object's id.
fRadiusFeet The radius in feet.
pfCallback A pointer to the callback interface to be registered.

Returns

S_OK if the environment force notification was successfully registered, E_FAIL otherwise.

◆ UnregisterEnvironmentForce()

virtual HRESULT
UnregisterEnvironmentForce (__in __notnull IEnvironmentForceV400 * pEnvironmentForce) private virtual

Unregisters a developer defined environment force.

Parameters

pEnvironmentForce The environment force to be destroyed.

Returns

S_OK if the environment force was successfully unregistered, E_FAIL otherwise.

◆ UnregisterNotification()

virtual HRESULT
UnregisterNotification (__in UINT uObjectId,
__in __notnull IEnvironmentForceCallbackV410 * pfCallback
)

private virtual

Unregisters an environment force notification callback.

Parameters

uObjectId The object's id.
pfCallback A pointer to the callback interface to be unregistered.

Returns
S_OK if the environment force notification was successfully unregistered, E_FAIL otherwise.

◆ P3D::IEnvironmentForceV400

class P3D::IEnvironmentForceV400

This interface may be implemented by developers to register environment forces.

Inherits IUnknown.

Private Member Functions

```
virtual float GetRadius() const PURE  
virtual float GetHalfAngle() const PURE  
virtual void GetPosition(__out DXYZ &vWorldPosRadiansFeet) const PURE  
virtual void GetForce(__out DXYZ &vWorldForcePounds) const PURE  
virtual void GetForceAtLocation(__in const DXYZ &vWorldPosRadiansFeet, __out DXYZ &vWorldForcePounds) const PURE  
virtual UINT GetOwnerId() const PURE
```

Member Function Documentation

◆ GetForce()

virtual void GetForce(__out DXYZ & vWorldForcePounds) const [private] [virtual]

The world force at the environment force location.

Parameters

[out] vWorldForcePounds The world force in pounds.

◆ GetForceAtLocation()

```
virtual void GetForceAtLocation(__in const DXYZ & vWorldPosRadiansFeet,  
                                __out DXYZ & vWorldForcePounds  
                                ) const [private] [virtual]
```

The world force at the given world position.

Parameters

vWorldPosRadiansFeet The world location in radians/feet.
[out] vWorldForcePounds The world force in pounds.

◆ GetHalfAngle()

virtual float GetHalfAngle() const [private] [virtual]

A half-angle value used to further define the volume of the environment force.

Returns

The half-angle of the volume in radians.

Remarks

Examples: PI / 4 would define a cone. PI / 2 would define a half-sphere. PI would define a complete sphere.

◆ GetOwnerId()

virtual UINT GetOwnerId() const [private] [virtual]

The object id of the environment force owner.

Returns

The owner's object id (zero for no owner).

◆ GetPosition()

virtual void GetPosition(__out DXYZ & vWorldPosRadiansFeet) const [private] [virtual]

The world location of the environment force.

Parameters

[out] vWorldPosRadiansFeet The world location in radians/feet.

◆ GetRadius()

```
virtual float GetRadius( ) const private virtual
```

The effective radius of the environment force.

Returns

The radius in feet.

Typedefs

```
using IEnvironmentForceList = IListBuilder< IEnvironmentForceV400 >
```

```
using EnvironmentForceList = CComPtrVecBuilder< IEnvironmentForceV400 >
```

Variables

```
GUID IID_IEnvironmentForceManagerV410
```

```
GUID SID_EnvironmentForceManager
```

```
GUID IID_IEnvironmentForceV400
```

```
GUID SID_EnvironmentForce
```

Typedef Documentation

◆ EnvironmentForceList

```
using EnvironmentForceList = CComPtrVecBuilder< IEnvironmentForceV400 >
```

◆ IEnvironmentForceList

```
using IEnvironmentForceList = IListBuilder< IEnvironmentForceV400 >
```

Variable Documentation

◆ IID_IEnvironmentForceManagerV410

```
GUID IID_IEnvironmentForceManagerV410
```

◆ IID_IEnvironmentForceV400

```
GUID IID_IEnvironmentForceV400
```

◆ SID_EnvironmentForce

```
GUID SID_EnvironmentForce
```

◆ SID_EnvironmentForceManager

```
GUID SID_EnvironmentForceManager
```

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Library Object Data

Overview

The **ILibraryObjectData** interface provides capabilities associated with library objects.

An **ILibraryObjectData** pointer can be obtained from the **IPdk QueryService()** method.

Classes

class **ILibraryObjectV400**

Class Documentation

◆ P3D::ILibraryObjectV400

class P3D::ILibraryObjectV400

ILibraryObject Provides data and access on a library object.

Inherits **IUnknown**.

Private Member Functions

virtual **UINT** **GetId** () const **PURE**

virtual **LPCTSTR** **GetDescription** () const **PURE**

virtual **HRESULT** **GetLonAltLat** (**_out** **P3DDXYZ** &vLonAltLat) const **PURE**

virtual **HRESULT** **AttachService** (**_in** **REFGUID** guidService, **_in** **IUnknown** *pUnkService)
PURE

virtual **HRESULT** **DetachService** (**_in** **REFGUID** guidService) **PURE**

Member Function Documentation

◆ AttachService()

```
virtual HRESULT AttachService ( __in REFGUID guidService,  
                               __in IUnknown * pUnkService  
                           )
```

private virtual

Enables an IUnknown-based service to be attached to this object. This is meant for ownership and storage only. It is up to the developer to implement and execute any runtime methods.

Parameters

Guid for the specific service.
pUnkService void pointer to a service to be attached.

Returns

Returns S_OK if the service was attached appropriately.

◆ DetachService()

```
virtual HRESULT DetachService ( __in REFGUID guidService )
```

private virtual

Removes an IUnknown-based interface to be attached to this object.

Parameters

Guid for the specific service.

Returns

Returns S_OK if the service was found and detached successfully

◆ GetDescription()

```
virtual LPCTSTR GetDescription ( ) const
```

private virtual

Returns

Returns the description of the object. When adding an object in SimDirector, this will be the "Object Name".

◆ GetId()

```
virtual UINT GetId ( ) const
```

private virtual

Returns

Returns the Prepar3D unique ID for this object. -1 is an invalid ID.

◆ GetLonAltLat()

```
virtual HRESULT GetLonAltLat ( __out P3DDXYZ & vLonAltLat ) const
```

private virtual

Query for the world position of the object.

Parameters

vLonAltLat X = Longitude in radians, Y = altitude in feet, Z = Latitude in radians.

Returns

Returns S_OK if the query returns valid data.

Variables

GUID IID_IListableObjectV400

Current version of IListableObject interface. [More...](#)

Variable Documentation

◆ IID_IListableObjectV400

GUID IID_IListableObjectV400

Current version of IListableObject interface.

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Icon Service

Overview

This PDK service allows callers to manipulate icons during runtime. Icons registered and unregistered using this interface. Icons utilize world-based positioning and may or may not be associated with an object.

Classes

- class [IIIconServiceV410](#)
- class [IIIconInstanceV410](#)

Class Documentation

◆ [P3D::IIIconServiceV410](#)

class P3D::IIIconServiceV410

This service is responsible for the management of Icons.

Inherits IUnknown.

Private Member Functions

- virtual **UINT32** [GetIconInstanceCount \(\) const PURE](#)
- virtual void [GetIconInstanceList \(_out IIIconInstanceList &pList\) PURE](#)
- virtual [IIIconInstanceV410 * CreateIconInstance \(\) PURE](#)
- virtual **HRESULT** [Register \(_in IIIconInstanceV410 *pIconInstance\) PURE](#)
- virtual **HRESULT** [Unregister \(_in IIIconInstanceV410 *pIconInstance\) PURE](#)

Member Function Documentation

◆ CreateIconInstance()

virtual **IIIconInstanceV410*** CreateIconInstance() **private** **virtual**

Creates an icon instances.

Returns

An Icon Instance.

◆ GetIconInstanceCount()

virtual **UINT32** GetIconInstanceCount() **const** **private** **virtual**

Returns the number of icon instances.

Returns

The number of icon instances.

◆ GetIconInstanceList()

virtual void GetIconInstanceList(__out **IIIconInstanceList** & pList) **private** **virtual**

Returns a list of icon instances.

Parameters

pList The list of Icon Instances.

◆ Register()

virtual **HRESULT** Register(__in **IIIconInstanceV410** * plconInstance) **private** **virtual**

Registers a developer defined icon instance.

Parameters

plconInstance The **IIIconInstanceV410** to be registered.

Returns

S_OK if the icon instance was successfully registered, **E_FAIL** otherwise.

◆ Unregister()

virtual **HRESULT** Unregister(__in **IIIconInstanceV410** * plconInstance) **private** **virtual**

Unregisters a developer defined icon instance.

Parameters

plconInstance The icon instance to be destroyed.

Returns

S_OK if the icon instance was successfully unregistered, E_FAIL otherwise.

◆ P3D::IIconInstanceV410

class P3D::IIconInstanceV410

This interface may be used by developers to manipulate icon instances.

Inherits IUnknown.

Private Member Functions

```
virtual void GetPosition ( __out DXYZ &vWorldPosRadiansFeet) const PURE
virtual void SetPosition ( __in const DXYZ &vWorldPosRadiansFeet) PURE
virtual UINT GetObjectId () const PURE
virtual HRESULT SetObjectId ( __in UINT objectId) PURE
virtual const GUID & GetTextureDefinition () const PURE
virtual HRESULT SetTextureDefinition ( __in const GUID &guid) PURE
virtual bool IsVisible () const PURE
virtual void SetVisibility ( __in bool visibility) PURE
```

Member Function Documentation

◆ GetObjectId()

virtual UINT GetObjectId () const private virtual

Attaches the position of a given icon instance to an object.

Parameters

objectId The object's id.

Returns

Will return 0 if there is no object id, otherwise it will return object's id.

◆ GetPosition()

virtual void GetPosition (__out DXYZ & vWorldPosRadiansFeet) const private virtual

Returns the position of a given icon instance.

Parameters

vWorldPosRadiansFeet The world location in radians/feet.

◆ GetTextureDefinition()

virtual const GUID& GetTextureDefinition() const **private** **virtual**

Sets the texture of a given icon instance.

Parameters

guid The icon/texture guid.

Returns

Returns the guid texture reference.

Remarks

The GUID mapping is set in the IconConfiguration.xml file.

◆ IsVisible()

virtual bool IsVisible() const **private** **virtual**

Returns visibility status of a given icon instance.

Returns

true if visibility is set to true, false if visibility is set to false.

Remarks

visibility = true draws the icon, visibility = false doesn't draw the icon.

◆ SetObjectId()

virtual HRESULT SetObjectId(__in UINT **objectId**) **private** **virtual**

Attaches the position of a given icon instance to an object.

Parameters

objectId The object's id.

Returns

S_OK if the icon instance was successfully set to an object, E_FAIL otherwise.

Remarks

An objectId of 0 will detach the icon instance from an object. SetObject will override SetPosition until SetObject(0) is called.

◆ SetPosition()

virtual void SetPosition(__in const **DXYZ** & **vWorldPosRadiansFeet**) **private** **virtual**

Sets the position of a given icon instance.

Parameters

vWorldPosRadiansFeet The world location in radians/feet.

◆ **SetTextureDefinition()**

virtual HRESULT SetTextureDefinition (__in const GUID & **guid**) **private** **virtual**

Sets the texture of a given icon instance.

Parameters

guid The icon/texture guid.

Returns

S_OK if the guid was successfully changed, E_FAIL otherwise.

Remarks

The GUID mapping is set in the IconConfiguration.xml file.

◆ **SetVisibility()**

virtual void SetVisibility (__in bool **visibility**) **private** **virtual**

Sets visibility status of a given icon instance.

Parameters

visibility The visibility flag.

Remarks

visibility = true draws the icon, visibility = false doesn't draw the icon.

Typedefs

using **IIconInstanceList** = **IListBuilder< IIconInstanceV410 >**

using **IconInstanceList** = **CComPtrVecBuilder< IIconInstanceV410 >**

Variables

GUID **IID_IIconServiceV410**

GUID **SID_IIconService**

GUID **IID_IIconInstanceV410**

GUID **SID_IIconInstance**

Typedef Documentation

◆ **IconInstanceList**

using **IconInstanceList** = **CComPtrVecBuilder< IIconInstanceV410 >**

◆ **IIconInstanceList**

using **IIconInstanceList = IListBuilder<IIconInstanceV410>**

Variable Documentation

◆ **IID_IIconInstanceV410**

GUID IID_IIconInstanceV410

◆ **IID_IIconServiceV410**

GUID IID_IIconServiceV410

◆ **SID_IIconInstance**

GUID SID_IIconInstance

◆ **SID_IIconService**

GUID SID_IIconService

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Voice Control Service

Overview

This service provides access to Prepar3D's voice control system. It can be used to add custom voice controls to the application.

Classes

class **IVoiceControlServiceV440**

Class Documentation

◆ **P3D::IVoiceControlServiceV440**

class P3D::IVoiceControlServiceV440

This service provides access to Prepar3D's voice control system.

Inherits [IVoiceControlServiceV420](#).

Private Member Functions

virtual void	EnableVoiceRecognition ()	PURE
virtual void	DisableVoiceRecognition ()	PURE
virtual bool	IsVoiceRecognitionEnabled ()	PURE
virtual int	AddPhrase (LPCWSTR phrase)	PURE
virtual int	AddPhrase (LPCWSTR phrase, double certainty)	PURE
virtual int	AddPhrase (LPCWSTR phrase, const wchar_t *event)	PURE
virtual int	AddPhrase (LPCWSTR phrase, const wchar_t *event, double certainty)	PURE
virtual void	RemovePhrase (LPCWSTR phrase)	PURE
virtual void	ClearPhrases ()	PURE

```
virtual void GetVoiceDebugInfo () PURE
virtual bool GetEnableVoiceDebug () PURE
virtual void SetEnableVoiceDebug (bool isEnabled) PURE
virtual void ReloadGrammar () PURE
virtual void RegisterPhraseCallback (ICallbackV400 *pCallback) PURE
virtual void UnregisterPhraseCallback (ICallbackV400 *pCallback) PURE
```

Member Function Documentation

◆ AddPhrase() [1 / 4]

```
virtual int AddPhrase (LPCWSTR phrase) [private] [virtual]
```

Adds a phrase from the language. ReloadPhrases will need to be called for changes to take effect.

Parameters

- phrase** The voice command to add.
- event** Optional string of the key event to bind the phrase to.
- certainty** Minimum certainty required for voice command to trigger (0.0 - 1.0). Defaults to 0.9

Returns

ID generated unique to this phrase. If no event is provided, this ID will be signalled to Callbacks when triggered.

◆ AddPhrase() [2 / 4]

```
virtual int AddPhrase (LPCWSTR phrase,
                      double certainty)
) [private] [virtual]
```

◆ AddPhrase() [3 / 4]

```
virtual int AddPhrase (LPCWSTR phrase,
                      const wchar_t * event)
) [private] [virtual]
```

◆ AddPhrase() [4 / 4]

```
virtual int AddPhrase (LPCWSTR phrase,
                      const wchar_t * event,
                      double certainty)
) [private] [virtual]
```

◆ ClearPhrases()

virtual void ClearPhrases() private virtual

Clears all phrases loaded into voice controls (including the default phrases).

◆ DisableVoiceRecognition()

virtual void DisableVoiceRecognition() private virtual

Disables voice recognition.

◆ EnableVoiceRecognition()

virtual void EnableVoiceRecognition() private virtual

Enables voice recognition.

◆ GetEnableVoiceDebug()

virtual bool GetEnableVoiceDebug() private virtual

Get whether or not to draw debug text.

◆ GetVoiceDebugInfo()

virtual wchar_t* GetVoiceDebugInfo() private virtual

Get Voice debug info. This is currently the heard phrase, confidence level

◆ IsVoiceRecognitionEnabled()

virtual bool IsVoiceRecognitionEnabled() private virtual

Check if voice recognition is enabled.

Returns

true if voice recognition is enabled, false otherwise.

◆ RegisterPhraseCallback()

virtual void RegisterPhraseCallback(**ICallbackV400** * pCallback) private virtual

Register a phrase callback.

Parameters

pCallback Callback to register

◆ ReloadGrammar()

virtual void ReloadGrammar () **private** **virtual**

Reloads the phrases with phrases added and removed since last load.

◆ RemovePhrase()

virtual void RemovePhrase (LPCWSTR **phrase**) **private** **virtual**

Removes a specific phrase from the language. ReloadPhrases will need to be called for changes to take effect.

Parameters

phrase The voice command to remove.

◆ SetEnableVoiceDebug()

virtual void SetEnableVoiceDebug (bool **isEnabled**) **private** **virtual**

Set whether or not to draw debug text.

◆ UnregisterPhraseCallback()

virtual void UnregisterPhraseCallback (**ICallbackV400** * **pCallback**) **private** **virtual**

Unregister a phrase callback.

Parameters

pCallback Callback to unregister

Variables

REFIID **IID_IVoiceControlServiceV440** = __uuidof(**IVoiceControlServiceV440**)

REFIID **SID_VoiceControlService** = **IID_IVoiceControlServiceV440**

Variable Documentation

◆ **IID_IVoiceControlServiceV440**

REFIID IID_IVoiceControlServiceV440 = __uuidof(**IVoiceControlServiceV440**)

◆ **SID_VoiceControlService**

REFIID SID_VoiceControlService = **IID_IVoiceControlServiceV440**

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

IEngineSystem

Overview

This interface can be used to implement a custom engine/propulsion system plugin, and still allow other core aircraft subsystems to interact appropriately with the custom engine system. A sample implementation using this interface can be found in the MyEngine sample in the PDK General Samples.

Remarks

- Engine indexes are 0-based.
- To enable a custom engine system plugin on a native aircraft, set in the aircraft.cfg or sim.cfg:
[GeneralEngineData]
EngineSystem = {bc95b363-1d22-42aa-82b1-f10905b22c40}
engine_type = 2

Where,

- engine type = 2 indicates to **P3D** to not create any core engine implementations
- The EngineSystem guid must match the service ID (SID_) of the subsystem factory registered at DLL load with the PDK. See the MyEngine sample.

Classes

class **IEngineSystemV500**

class **ISavedEngineSystemStateV500**

Class Documentation

◆ **P3D::IEngineSystemV500**

class P3D::IEngineSystemV500

IEngineSystem is the primary interface for the core **P3D** aircraft simulation to get and set data inside the engine system implementation.

Inherits **ISimulationV310**.

Private Member Functions

virtual HRESULT	SaveState (_out ISavedEngineSystemStateV500 **ppSavedState) const PURE
virtual HRESULT	LoadState (_in const ISavedEngineSystemStateV500 *pSavedState) PURE
virtual HRESULT	Init () PURE
virtual HRESULT	DeInit () PURE
virtual UINT	GetNumberOfEngines () const PURE
virtual UINT	GetNumberOfPropellers () const PURE
virtual double	GetMaxRpm () const PURE
virtual HRESULT	SetPowerOff (BOOL bSetAllEng= TRUE , UINT iEng=0) PURE
virtual HRESULT	SetPowerIdle (BOOL bSetAllEng= TRUE , UINT iEng=0) PURE
virtual HRESULT	SetPowerCruise (BOOL bSetAllEng= TRUE , UINT iEng=0) PURE
virtual HRESULT	ResetDamage () PURE
virtual HRESULT	SetFailed (BOOL bFailed, UINT iEng) PURE
virtual BOOL	IsCombusting (UINT iEng) const PURE
virtual double	GetThrottle (UINT iEng) const PURE
virtual HRESULT	SetThrottle (double dThrottle, UINT iEng) PURE
virtual double	GetPercentRpm (UINT iEng) const PURE
virtual HRESULT	SetPercentRpm (double dPctRpm, UINT iEng) PURE
virtual HRESULT	GetOffset (_out FXYZ &vOffset, UINT iEng) const PURE
virtual HRESULT	GetAngle (_out FXYZ &vAngle, UINT iEng) const PURE
virtual double	GetThrust (UINT iEng) const PURE
virtual double	GetShaftTorque (UINT iEng) const PURE
virtual double	GetFuelFlowPPH (UINT iEng) const PURE
virtual int	GetFuelTankSelector (UINT iEng) const PURE
virtual HRESULT	SetFuelTankSelector (int eSelector, UINT iEng) PURE
virtual HRESULT	SetFuelTanksUsed (_in BOOL bFuelAvailable, _in int bfTanksUsed, _in UINT nTanksUsed, _in UINT iEng) PURE
virtual HRESULT	GetFuelTanksUsed (_out int &bfTanksUsed, _out UINT &nTanksUsed, _in UINT iEng) const PURE
virtual double	GetTurbineN1 (UINT iEng) const PURE
virtual double	GetTurbineBleedAirPressure (UINT iEng) const PURE
virtual double	GetTurbinePercentTorque (UINT iEng) const PURE
virtual double	GetRecipManifoldPressure (UINT iEng) const PURE
virtual double	GetPropellerPercentRpm (UINT iProp) const PURE
virtual double	GetPropellerAbsorbedTorque (UINT iProp) const PURE

virtual double	GetPropellerAbsorbedTorque (UINT iProp) const PURE
virtual double	GetPropellerThrust (UINT iProp) const PURE
virtual double	GetPropellerInducedVelocity (UINT iProp) const PURE
virtual HRESULT	GetPropellerMoment (<u>out</u> DXYZ &vMoment, UINT iProp) const PURE
virtual HRESULT	GetPropellerOffset (<u>out</u> FXYZ &vOffset, UINT iProp) const PURE

Member Function Documentation

◆ DeInit()

virtual HRESULT Delnit() private virtual

Called on aircraft de-initialization.

◆ GetAngle()

Returns the angle (in radians) of the engine's thrust from the aircraft's X-Z axes.

Banks angle (Z-component) is typically expected to be zero.

Parameters

vAngle Out-parameter set to the total angle vector.

◆ GetFuelFlowPPH()

virtual double GetFuelFlowPPH (UINT iEng) const **private** **virtual**

Returns the engine fuel flow rate (pounds per hour).

◆ GetFuelTankSelector()

virtual int GetFuelTankSelector (UINT iEng) const **private** **virtual**

Returns the fuel selector ID value for the engine. Refer to the Fuel Tank Selection list in Simulation Variables for a complete list of IDs.

◆ GetFuelTanksUsed()

```
    __in UINT      iEng  
    )          const
```

```
private virtual
```

Returns a bit-field of tanks currently being used and the total count of tanks used. It is the engine system's responsibility to maintain this data.

Parameters

bfTanksUsed Bit-field indicating which specific tanks are feeding the engine. The complete list of individual tank masks can be found in Simulation Variables.

nTanksUsed Total number of tanks currently feeding the engine.

◆ GetMaxRpm()

```
virtual double GetMaxRpm( ) const private virtual
```

Returns the maximum rated RPM for the engines.

◆ GetNumberOfEngines()

```
virtual UINT GetNumberOfEngines( ) const private virtual
```

Returns total number of engines.

◆ GetNumberOfPropellers()

```
virtual UINT GetNumberOfPropellers( ) const private virtual
```

Returns total number of propellers.

◆ GetOffset()

```
virtual HRESULT GetOffset( __out FXYZ & vOffset,  
                           UINT           iEng  
                         )          const private virtual
```

Returns the offset (in feet) of the engine from the aircraft's datum.

Parameters

vOffset Out-parameter set to the total offset vector.

◆ GetPercentRpm()

```
virtual double GetPercentRpm( UINT iEng ) const private virtual
```

Returns the engine RPM percentage of maximum rated RPM, typically 0.0 - 1.0.

◆ GetPropellerAbsorbedTorque()

```
virtual double GetPropellerAbsorbedTorque ( UINT iProp ) const [private] [virtual]
```

Returns for a propeller the torque required, or absorbed, for its current speed and atmospheric conditions (foot-pounds).

◆ GetPropellerInducedVelocity()

```
virtual double GetPropellerInducedVelocity ( UINT iProp ) const [private] [virtual]
```

Returns for a propeller the induced velocity currently being generated (feet per second). This is used on center-line thrust propeller aircraft to generate additional forces on tail surfaces.

◆ GetPropellerMoment()

```
virtual HRESULT GetPropellerMoment ( __out DXYZ & vMoment,  
                                     UINT          iProp  
                               )           const [private] [virtual]
```

Returns additional moments generated by the propeller (foot-pounds). Examples of effects contributing to this are gyroscopic effects and P-factor. This should not include moments due to the propellers offset from the center-of-gravity.

Parameters

vMoment Out-parameter set to the total moment vector.

◆ GetPropellerOffset()

```
virtual HRESULT GetPropellerOffset ( __out FXYZ & vOffset,  
                                    UINT          iProp  
                               )           const [private] [virtual]
```

Returns the propeller offset from the aircraft's datum (feet).

Parameters

vOffset Out-parameter set to the total offset vector.

◆ GetPropellerPercentRpm()

```
virtual double GetPropellerPercentRpm ( UINT iProp ) const [private] [virtual]
```

Returns for a propeller the percentage of maximum rated RPM (typically 0.0 - 1.0).

◆ GetPropellerThrust()

```
virtual double GetPropellerThrust ( UINT iProp ) const [private] [virtual]
```

Returns for a propeller the thrust scalar being generated (pounds-force).

◆ GetRecipManifoldPressure()

```
virtual double GetRecipManifoldPressure ( UINT iEng ) const [private] [virtual]
```

Returns for a reciprocating engine manifold pressure (pounds per square foot).

◆ GetShaftTorque()

```
virtual double GetShaftTorque ( UINT iEng ) const [private] [virtual]
```

Returns the engine shaft torque (foot-pounds).

◆ GetThrottle()

```
virtual double GetThrottle ( UINT iEng ) const [private] [virtual]
```

Returns engine throttle percentage, typically 0.0 - 1.0.

◆ GetThrust()

```
virtual double GetThrust ( UINT iEng ) const [private] [virtual]
```

Returns the thrust scalar value (pounds-force).

◆ GetTurbineBleedAirPressure()

```
virtual double GetTurbineBleedAirPressure ( UINT iEng ) const [private] [virtual]
```

Returns for a turbine engine the Bleed Air Pressure (pounds per square foot).

◆ GetTurbineN1()

```
virtual double GetTurbineN1 ( UINT iEng ) const [private] [virtual]
```

Returns for a turbine engine the N1 engine speed. Typically 0 - 100.

◆ GetTurbinePercentTorque()

virtual double GetTurbinePercentTorque (**UINT** iEng) const **private** **virtual**

Returns for a turbine engine the percentage of maximum rated torque (typically 0.0 - 1.0).

◆ Init()

virtual **HRESULT** Init () **private** **virtual**

Called on aircraft initialization.

◆ IsCombusting()

virtual **BOOL** IsCombusting (**UINT** iEng) const **private** **virtual**

Returns engine combustion state

◆ LoadState()

virtual
HRESULT
LoadState (**__in const ISavedEngineSystemStateV500** * pSavedState) **private** **virtual**

Passes a reference to a saved class holding the the state to initialize the engine system. This is used for scenario loading and state transfer when switching aircraft.

Parameters

pSavedState Address of a object holding the current state.

◆ ResetDamage()

virtual **HRESULT** ResetDamage () **private** **virtual**

Resets all damage states on the engines.

◆ SaveState()

virtual
HRESULT
SaveState (**__out ISavedEngineSystemStateV500** ** ppSavedState) const **private** **virtual**

Returns a reference to a saved class holding the current state of the engine system. This is used for scenario saving and state transfer when switching aircraft.

Parameters

ppSavedState Address of a object pointer holding the current state.

◆ SetFailed()

```
virtual HRESULT SetFailed ( BOOL bFailed,  
                           UINT iEng  
                         )
```

private **virtual**

Set engine failed state.

◆ SetFuelTankSelector()

```
virtual HRESULT SetFuelTankSelector ( int eSelector,  
                                      UINT iEng  
                                    )
```

private **virtual**

Sets the fuel selector ID value for the engine. Refer to the Fuel Tank Selection list in Simulation Variables for a complete list of IDs.

◆ SetFuelTanksUsed()

```
virtual HRESULT SetFuelTanksUsed ( __in BOOL bFuelAvailable,  
                                 __in int bfTanksUsed,  
                                 __in UINT nTanksUsed,  
                                 __in UINT iEng  
                               )
```

private **virtual**

Called by the fuel system with a bit-field of tanks currently being used. It is the engine system's responsibility to maintain this data.

Parameters

bFuelAvailable Indicates if the fuel system is providing available fuel to the engine, based on the engine's fuel selector.

bfTanksUsed Bit-field indicating which specific tanks are feeding the engine. The complete list of individual tank masks can be found in Simulation Variables.

nTanksUsed Total number of tanks currently feeding the engine.

◆ SetPercentRpm()

```
virtual HRESULT SetPercentRpm ( double dPctRpm,  
                               UINT iEng  
                             )
```

private **virtual**

Sets the engine RPM percentage of maximum rated RPM, typically 0.0 - 1.0. Typically used for initialization, as well as certain transmission feedback scenarios on some aircraft.

◆ SetPowerCruise()

```
virtual HRESULT SetPowerCruise ( BOOL bSetAllEng = TRUE,  
                                UINT iEng = 0  
                            )
```

private virtual

Request to set engine power to a cruise setting. Generally used for initializations such as repositioning from the UI.

Parameters

bSetAllEng TRUE dictates all engines should be set. If FALSE, set the engine indicated by iEng.

iEng Indicates which engine to set. Ignored if bSetAllEng = TRUE.

◆ SetPowerIdle()

```
virtual HRESULT SetPowerIdle ( BOOL bSetAllEng = TRUE,  
                             UINT iEng = 0  
                           )
```

private virtual

Request to set engine power to idle. Generally used for initializations such as repositioning from the UI.

Parameters

bSetAllEng TRUE dictates all engines should be set. If FALSE, set the engine indicated by iEng.

iEng Indicates which engine to set. Ignored if bSetAllEng = TRUE.

◆ SetPowerOff()

```
virtual HRESULT SetPowerOff ( BOOL bSetAllEng = TRUE,  
                            UINT iEng = 0  
                          )
```

private virtual

Request to set engine power off. Generally used for initializations such as repositioning from the UI.

Parameters

bSetAllEng TRUE dictates all engines should be set. If FALSE, set the engine indicated by iEng.

iEng Indicates which engine to set. Ignored if bSetAllEng = TRUE.

◆ SetThrottle()

```
virtual HRESULT SetThrottle ( double dThrottle,  
                           UINT iEng  
                         )
```

private virtual

Sets throttle percentage, typically 0.0 - 1.0. This would be used for both initialization as well as autopilot throttle controllers.

◆ P3D::ISavedEngineSystemStateV500

class P3D::ISavedEngineSystemStateV500

ISavedEngineSystemState is used during loading and saving the state of the engine system when saving and loading scenarios, as well as when switching aircraft.

Inherits IUnknown.

Private Member Functions

```
virtual UINT GetNumberOfEngines () const PURE  
virtual BOOL GetCombustion (UINT iEng) const PURE  
virtual float GetPercentEngineRpm (UINT iEng) const PURE  
virtual float GetThrottleLeverPosition (UINT iEng) const PURE  
virtual float GetPropellerLeverPosition (UINT iEng) const PURE  
virtual float GetMixtureLeverPosition (UINT iEng) const PURE
```

Member Function Documentation

◆ GetCombustion()

```
virtual BOOL GetCombustion ( UINT iEng ) const private virtual
```

Returns TRUE if the specified engine is combusting.

◆ GetMixtureLeverPosition()

```
virtual float GetMixtureLeverPosition ( UINT iEng ) const private virtual
```

Returns the mixture (or condition) lever percentage for the specified engine. Typically 0.0 - 1.0.

◆ GetNumberOfEngines()

```
virtual UINT GetNumberOfEngines ( ) const private virtual
```

Returns the number of engines on this aircraft.

◆ **GetPercentEngineRpm()**

virtual float GetPercentEngineRpm(**UINT iEng**) const **private virtual**

Returns the percentage of maximum rated RPM for the specified engines. Typically 0.0 - 1.0.

◆ **GetPropellerLeverPosition()**

virtual float GetPropellerLeverPosition(**UINT iEng**) const **private virtual**

Returns the propeller lever percentage for the specified engine. Typically 0.0 - 1.0.

◆ **GetThrottleLeverPosition()**

virtual float GetThrottleLeverPosition(**UINT iEng**) const **private virtual**

Returns the throttle percentage for the specified engine. Typically 0.0 - 1.0.

Variables

GUID IID_IEngineSystemV500

Current versioned ID of the Engine System interface. [More...](#)

GUID SID_EngineSystem

Service ID for the engine system. [More...](#)

GUID IID_ISavedEngineSystemStateV500

Current versioned ID of the Saved Engine State interface. [More...](#)

Variable Documentation

◆ **IID_IEngineSystemV500**

GUID IID_IEngineSystemV500

Current versioned ID of the Engine System interface.

◆ **IID_ISavedEngineSystemStateV500**

GUID IID_ISavedEngineSystemStateV500

Current versioned ID of the Saved Engine State interface.

◆ SID_EngineSystem

GUID SID_EngineSystem

Service ID for the engine system.

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Scenario Manager Service

Overview

This PDK service provides the current scenario (also known as "flight") and mission files.

Namespaces

P3D

Prepar3D SDK namespace used primarily for the PDK and its services.

Classes

class **IScenarioManagerV453**

Class Documentation

◆ P3D::IScenarioManagerV453

class P3D::IScenarioManagerV453

Provides the current scenario (also known as "flight") and mission files.

Inherits IScenarioManagerV430.

Private Member Functions

```
virtual HRESULT GetScenarioFilePath ( __out LPWSTR pszFilePath, __in UINT uLength) const PURE
virtual HRESULT GetObjectFilePath ( __out LPWSTR pszFilePath, __in UINT uLength) const PURE
virtual HRESULT GetObjectName ( __out LPWSTR pszFileName, __in UINT uLength) const PURE
virtual HRESULT GetScenarioFilePathList ( __out INameList &names) const PURE
virtual HRESULT LoadScenario ( __in LPWSTR pszFilePath) const PURE
virtual HRESULT GetMissionObjectiveList ( __out IMissionObjectiveList &missionObjectives) const PURE
virtual HRESULT GetGoalList ( __out IGoalList &goals) const PURE
virtual HRESULT GetFlightSegmentList ( __out IFlightSegmentList &flightSegments) const PURE
virtual HRESULT RegisterForNewScenarioNotification ( __in PNewScenarioNotify
pcbNotifyFunction) PURE
```

Member Function Documentation

◆ GetFlightSegmentList()

virtual HRESULT GetFlightSegmentList (__out IFlightSegmentList & flightSegments) const **private** **virtual**

Provides the list of flight segments.

Parameters
flightSegments

◆ GetGoalList()

virtual HRESULT GetGoalList (__out IGoalList & goals) const **private** **virtual**

Provides the list of goals for the scenario

Parameters

goals

◆ **GetMissionObjectiveList()**

```
virtual HRESULT GetMissionObjectiveList ( __out IMissionObjectiveList & missionObjectives ) const [private] [virtual]
```

Provides the list of mission objectives for the scenario

Parameters

missionObjectives

◆ **GetObjectFileName()**

```
virtual HRESULT GetObjectFileName ( __out LPWSTR pszFileName,
                                   __in  UINT      uLength
                               ) const [private] [virtual]
```

Provides the file name of the associated object file. If one does not exist a zero-length string is returned.

Parameters

pszFileName
 uLength

◆ **GetObjectFilePath()**

```
virtual HRESULT GetObjectFilePath ( __out LPWSTR pszFilePath,
                                   __in  UINT      uLength
                               ) const [private] [virtual]
```

Provides the fully qualified path to the associated object file. If one does not exist a zero-length string is returned.

Parameters

pszFilePath
 uLength

◆ **GetScenarioFilePath()**

```
virtual HRESULT GetScenarioFilePath ( __out LPWSTR pszFilePath,
                                       __in  UINT      uLength
                               ) const [private] [virtual]
```

Provides the fully qualified path to the scenario file.

Parameters

pszFilePath
 uLength

◆ **GetScenarioFilePathList()**

```
virtual HRESULT GetScenarioFilePathList ( __out INameList & names ) const [private] [virtual]
```

Provides a list of scenario paths.

Parameters

names

◆ **LoadScenario()**

```
virtual HRESULT LoadScenario ( __in LPWSTR pszFilePath ) const [private] [virtual]
```

Loads the specified scenario.

Parameters

pszFilePath**◆ RegisterForNewScenarioNotification()**

virtual HRESULT

RegisterForNewScenarioNotification (__in PNewScenarioNotify pcbNotifyFunction) **private** **virtual**

This allows registering a callback function for notification when the current scenario has been loaded or changed. See [Types.h](#) for the function profile of PNewScenarioNotify.

Parameters**pcbNotifyFunction**

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Configuration Service

Overview

This service provides access to configuration data defined in SimProp XML files.

Classes

```
class IConfiguratioSetV440  
struct SimConfigurationDescriptionV440  
class IConfigurationServiceV440
```

Class Documentation

◆ P3D::IConfiguratioSetV440

```
class P3D::IConfiguratioSetV440
```

A set of configuration data defined in a SimProp XML file

Inherits IUnknown.

Private Member Functions

```
virtual ISimPropertySet * GetPropSet () override  
virtual const GUID & GetGUID () const override  
virtual LPCWSTR GetName () const override  
virtual LPCWSTR GetFilePath () const  
virtual HRESULT SetPropSet (ISimPropertySet *pSet) override  
virtual HRESULT LoadFromFile () override  
virtual HRESULT SaveToFile () override  
virtual HRESULT RegisterCallback (ICallbackV100 *pCallback) override
```

virtual HRESULT RegisterCallback (ICallbackV400 *pCallback) override

virtual HRESULT UnregisterCallback (ICallbackV400 *pCallback) override

Member Function Documentation

◆ GetFilePath()

virtual LPCWSTR GetFilePath() const **private** **virtual**

Get the ISimPropertySet that holds the configuration data

Returns

LPCWSTR, the file path to the SimProp xml file containing the configuration data

◆ GetGUID()

virtual const GUID& GetGUID() const **private** **pure virtual**

Get the GUID (Globally Unique Identifier) for this configuration set

Returns

GUID, GUID of the configuration set

◆ GetName()

virtual LPCWSTR GetName() const **private** **pure virtual**

Get the name of this configuration set

Returns

LPCWSTR, same of the configuration set

◆ GetPropSet()

virtual **ISimPropertySet*** GetPropSet() **private** **pure virtual**

Get the ISimPropertySet that holds the configuration data

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ LoadFromFile()

virtual HRESULT LoadFromFile() **private** **pure virtual**

Load configuration data from file.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ RegisterCallback()

virtual HRESULT RegisterCallback (**ICallbackV400** * pCallback) **private** **pure virtual**

Register callback. Callbacks can be triggered when configuration data is changed, or when buttons are pressed in the configuration UI.

Parameters

pCallback Callback to register

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ SaveToFile()

virtual HRESULT SaveToFile () **private** **pure virtual**

Save configuration data to file.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ SetPropSet()

virtual HRESULT SetPropSet (**ISimPropertySet** * pSet) **private** **pure virtual**

Get the ISimPropertySet that holds the configuration data. This should trigger any callbacks listing for changes.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ UnregisterCallback()

virtual HRESULT UnregisterCallback (**ICallbackV400** * pCallback) **private** **pure virtual**

Unregister callback.

Parameters

pCallback Callback to unregister

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ P3D::SimConfigurationDescriptionV440

struct P3D::SimConfigurationDescriptionV440

Class Members

ICallbackV400 *	Callback	Callback to handle configuration changes or related UI events
LPCWSTR	FilePath	Path to XML file that contains the SimProp configuration data
GUID	ID	GUID ID or the configuration set
bool	IsNetworkSynchronized	Should this data be synchronized over the network. This allows setting changes on a multichannel host to be applied to all connected clients
LPCWSTR	Name	Name of the set. This will show up in the configuration UI
LPCWSTR	SymbolsPath	Path to the propdef XML file that defines the SimPropertySet schema

◆ P3D::IConfigurationServiceV440

class P3D::IConfigurationServiceV440

This service provides access configuration data stored in SimProp XML files

Inherits IUnknown.

Private Member Functions

virtual IConfigurationSetV440 * GetConfigurationSet (const GUID &cfgid)=0
virtual HRESULT GetConfigurationList (IConfigurationSetList &cfgList)=0
virtual HRESULT AddConfiguration (SimConfigurationDescriptionV440 &desc)=0
virtual HRESULT RemoveConfiguration (const GUID &cfgid)=0
virtual HRESULT ClearUISettingsList ()=0
virtual HRESULT AddConfigurationToUISettings (const GUID &cfgid)=0
virtual HRESULT SetUISettingsWindowTitle (const LPWSTR &windowTitle)=0
virtual HRESULT GetUIConfigurationList (IConfigurationSetList &cfgList)=0
virtual HRESULT OpenUISettingsWindow (bool bModal=true, UINT width=0, UINT height=0)=0
virtual HRESULT CloseUISettingsWindow ()=0
virtual HRESULT IsUISettingsWindowOpen (bool &bOpen)=0
virtual HRESULT AddUISettingsWindowButton (const LPWSTR &buttonName)=0
virtual HRESULT GetSelectedPropertyName (std::wstring &propertyName)=0

Member Function Documentation

◆ AddConfiguration()

virtual HRESULT AddConfiguration(SimConfigurationDescriptionV440 & desc) **private** **pure virtual**

Get list of all configuration sets.

Parameters

desc Description struct that defines the new set to be added.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ AddConfigurationToUISettings()

virtual HRESULT AddConfigurationToUISettings(const GUID & cfgid) **private** **pure virtual**

Add a configuration set to the UI settings window. This set will be editable via the UI after making a call to [OpenUISettingsWindow\(\)](#).

Parameters

cfgid ID of set to add.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ AddUISettingsWindowButton()

virtual HRESULT AddUISettingsWindowButton(const LPWSTR & buttonName) **private** **pure virtual**

Add button to settings window. Current max is 4. Button data is cleared when UI closes. When a button is pressed each configuration set in the UI will invoke its callbacks passing Button1Pressed, Button2Pressed, Button3Pressed, or Button4Pressed as the first callback parameter

Parameters

buttonName Text to show on the button

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ ClearUISettingsList()

virtual HRESULT ClearUISettingsList() **private** **pure virtual**

Clears the list of sets to be configured via the UI

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ CloseUISettingsWindow()

virtual HRESULT CloseUISettingsWindow() private pure virtual

Close UI settings window.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ GetConfigurationList()

virtual HRESULT
GetConfigurationList (IConfigurationSetList & cfgList) private pure virtual

Get list of all configuration sets.

Parameters

cfgList Empty list that will be filled with the requested data.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ GetConfigurationSet()

virtual IConfigurationSetV440*
GetConfigurationSet (const GUID & cfgid) private pure virtual

Get configuration set by ID @parm cfgid GUID id of configuration set

Returns

IConfigurationSetV440*, configuration set

◆ GetSelectedPropertyName()

virtual HRESULT
GetSelectedPropertyName (std::wstring & propertyName) private pure virtual

Get name of currently selected property in the settings UI

Parameters

[out] **propertyName** Name of selected property

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ GetUIConfigurationList()

virtual HRESULT
GetUIConfigurationList

(IConfigurationSetList & cfgList) **private** **pure virtual**

Get list of all configuration sets currently editable via the UI.

Parameters

cfgList Empty list that will be filled with the requested data.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ IsUISettingsWindowOpen()

virtual HRESULT IsUISettingsWindowOpen (bool & bOpen) **private** **pure virtual**

Check to see if UI window is open.

Parameters

bOpen Will be set to true if window is open, false if not

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ OpenUISettingsWindow()

virtual HRESULT OpenUISettingsWindow (bool bModal = `true`,
 UINT width = 0,
 UINT height = 0
)

private **pure virtual**

Open UI settings window.

Parameters

bModal Set this to true to make the window modal

width Width of UI window in pixels (Set 0 for default)

height Height of UI window in pixels (Set 0 for default)

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ RemoveConfiguration()

virtual HRESULT RemoveConfiguration (const GUID & cfgid) **private** **pure virtual**

Get list of all configuration sets.

Parameters

`cfgid` ID of set to remove.

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ SetUISettingsWindowTitle()

virtual HRESULT

SetUISettingsWindowTitle

(const LPWSTR & `windowTitle`) private pure virtual

Set the window title for the configuration UI. after making a call to [OpenUISettingsWindow\(\)](#).

Parameters

`windowTitle` Title text to show when in the settings UI window

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

Typedefs

typedef P3D::IListBuilder< IConfigurationSetV440 > IConfigurationSetList

typedef P3D::CComPtrVecBuilder< IConfigurationSetV440 > ConfigurationSetList

Enumerations

```
enum ConfigurationCallbackEvent : UINT64 {
    FileLoaded = 0, SimPropSetChanged = 1, FileSaved = 2, Button1Pressed = 3,
    Button2Pressed = 4, Button3Pressed = 5, Button4Pressed = 6, ExitDialog = 7
}
```

Variables

REFIID IID_IConfigurationServiceV440 = __uuidof(IConfigurationServiceV440)

REFIID IID_IConfigurationSetV440 = __uuidof(IConfigurationSetV440)

REFIID SID_ConfigurationService = IID_IConfigurationServiceV440

Typedef Documentation

◆ ConfigurationSetList

typedef P3D::CComPtrVecBuilder< IConfigurationSetV440 > ConfigurationSetList

Configuration list builder implementation using a standard vector of CComPtrs

◆ IConfigurationSetList

typedef P3D::IListBuilder< IConfigurationSetV440 > IConfigurationSetList

Configuration List Builder Interface

Enumeration Type Documentation

◆ ConfigurationCallbackEvent

enum **ConfigurationCallbackEvent** : **UINT64**

Enumerator

FileLoaded

SimPropSetChanged

FileSaved

Button1Pressed

Button2Pressed

Button3Pressed

Button4Pressed

ExitDialog

Variable Documentation

◆ IID_IConfigurationServiceV440

REFIID IID_IConfigurationServiceV440 = __uuidof(**IConfigurationServiceV440**)

◆ IID_IConfigurationSetV440

REFIID IID_IConfigurationSetV440 = __uuidof(**IConfigurationSetV440**)

◆ SID_ConfigurationService

REFIID SID_ConfigurationService = **IID_IConfigurationServiceV440**

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Controllable Camera

Overview

This service is provided through the PDK interface and provides a wrapper interface to control a camera using XML via a GaugeCallback plugin. While XML is the primary use case, a PDK service is provided to make it easier to extend the camera control capabilities via C++. The controllable camera service was developed entirely through the SDK to showcase the power and flexibility that 3rd party developers have.

Classes

class **IControllableCameraV450**

Class Documentation

◆ **P3D::IControllableCameraV450**

class P3D::IControllableCameraV450

Interface for ControllableCamera PDK service

Inherits [IControllableCameraV400](#).

Public Member Functions

```
virtual HRESULT Init (LPCWSTR sWindowName) override
virtual HRESULT DeInit (void) override
virtual bool IsInitialized () const override
virtual void SetVisible (bool visible) override
virtual bool IsVisible () override
virtual HRESULT Register () override
virtual bool IsRegistered () override
```

virtual void	ISystem () override
virtual void	SetPbh (float pitch, float bank, float heading) override
virtual void	SetPitch (float pitch) override
virtual void	SetBank (float bank) override
virtual void	SetHeading (float heading) override
virtual void	SetXyz (float x, float y, float z) override
virtual void	SetX (float x) override
virtual void	SetY (float y) override
virtual void	SetZ (float z) override
virtual void	ZoomIn () override
virtual void	ZoomOut () override
virtual void	SetZoom (float zoom) override
virtual void	SetFov (float hFov, float vFov) override
virtual void	SetHfov (float hFov) override
virtual void	SetVfov (float vFov) override
virtual void	ActivatePositionTracking () override
virtual void	DeactivatePositionTracking () override
virtual void	SetTargetLla (double lat, double lon, double alt) override
virtual void	SetTargetLat (double lat) override
virtual void	SetTargetLon (double lon) override
virtual void	SetTargetAlt (double alt) override
virtual void	ActivateEntityTracking () override
virtual void	DeactivateEntityTracking () override
virtual void	SetTargetContainerId (unsigned int targetId) override
virtual void	AddPostProcess (const WCHAR *postProcess) override
virtual void	RemovePostProcess (const WCHAR *postProcess) override
virtual void	ResetPostProcess () override
virtual void	ClearPostProcess () override
virtual void	SetSensorMode (unsigned int sensorMode) override
virtual void	SetContinuousPitch (float pitch) override
virtual void	SetContinuousBank (float bank) override
virtual void	SetContinuousHeading (float heading) override
virtual void	SetGlobalRotate (bool bGlobalRotate) override
virtual void	SetGlobalPbh (float pitch, float bank, float heading) override
virtual void	GetPbh (float &pitch, float &bank, float &heading) override
virtual float	GetPitch () override
virtual float	GetBank () override
virtual float	GetHeading () override
virtual void	GetXyz (float &x, float &y, float &z) override
virtual float	GetX () override

<code>virtual float GetY () override</code>
<code>virtual float GetZ () override</code>
<code>virtual float GetZoom () override</code>
<code>virtual void GetFov (float &hFov, float &vFov) override</code>
<code>virtual float GetHfov () override</code>
<code>virtual float GetVfov () override</code>
<code>virtual bool IsPositionTracking () override</code>
<code>virtual void GetTargetLla (double &lat, double &lon, double &alt) override</code>
<code>virtual double GetTargetLat () override</code>
<code>virtual double GetTargetLon () override</code>
<code>virtual double GetTargetAlt () override</code>
<code>virtual const WCHAR * GetTargetLatString () override</code>
<code>virtual const WCHAR * GetTargetLonString () override</code>
<code>virtual bool IsEntityTracking () override</code>
<code>virtual unsigned int GetTargetContainerId () override</code>
<code>virtual unsigned int GetSensorMode () override</code>
<code>virtual bool GetCameraLookAtLLA (double &lat, double &lon, double &alt) override</code>
<code>virtual bool IsGlobalRotate () override</code>
<code>virtual void GetGlobalPbh (float &pitch, float &bank, float &heading) override</code>
<code>virtual void ToggleSensorMode () override</code>
<code>virtual void ToggleTracking () override</code>
<code>virtual void TrackWeaponTarget () override</code>
<code>virtual void UntrackWeaponTarget () override</code>

Functions

<code>virtual HRESULT Init (LPCWSTR sWindowName) override</code>
<code>virtual HRESULT DeInit (void) override</code>
<code>virtual bool IsInitialized () const override</code>
<code>virtual void SetVisible (bool visible) override</code>
<code>virtual bool IsVisible () override</code>
<code>virtual HRESULT Register () override</code>
<code>virtual bool IsRegistered () override</code>
<code>virtual void SetPbh (float pitch, float bank, float heading) override</code>
<code>virtual void SetPitch (float pitch) override</code>
<code>virtual void SetBank (float bank) override</code>
<code>virtual void SetHeading (float heading) override</code>
<code>virtual void SetXyz (float x, float y, float z) override</code>
<code>virtual void SetX (float x) override</code>
<code>virtual void SetY (float y) override</code>

virtual void	SetX (float x) override
virtual void	SetZ (float z) override
virtual void	ZoomIn () override
virtual void	ZoomOut () override
virtual void	SetZoom (float zoom) override
virtual void	SetFov (float hFov, float vFov) override
virtual void	SetHfov (float hFov) override
virtual void	SetVfov (float vFov) override
virtual void	ActivatePositionTracking () override
virtual void	DeactivatePositionTracking () override
virtual void	SetTargetLla (double lat, double lon, double alt) override
virtual void	SetTargetLat (double lat) override
virtual void	SetTargetLon (double lon) override
virtual void	SetTargetAlt (double alt) override
virtual void	ActivateEntityTracking () override
virtual void	DeactivateEntityTracking () override
virtual void	SetTargetContainerId (unsigned int targetId) override
virtual void	AddPostProcess (const WCHAR *postProcess) override
virtual void	RemovePostProcess (const WCHAR *postProcess) override
virtual void	ResetPostProcess () override
virtual void	ClearPostProcess () override
virtual void	SetSensorMode (unsigned int sensorMode) override
virtual void	SetContinuousPitch (float pitch) override
virtual void	SetContinuousBank (float bank) override
virtual void	SetContinuousHeading (float heading) override
virtual void	SetGlobalRotate (bool bGlobalRotate) override
virtual void	SetGlobalPbh (float pitch, float bank, float heading) override
virtual void	GetPbh (float &pitch, float &bank, float &heading) override
virtual float	GetPitch () override
virtual float	GetBank () override
virtual float	GetHeading () override
virtual void	GetXyz (float &x, float &y, float &z) override
virtual float	GetX () override
virtual float	GetY () override
virtual float	GetZ () override
virtual float	GetZoom () override
virtual void	GetFov (float &hFov, float &vFov) override
virtual float	GetHfov () override
virtual float	GetVfov () override
virtual bool	IsPositionTracking () override

```
virtual void GetTargetLla (double &lat, double &lon, double &alt) override
virtual double GetTargetLat () override
virtual double GetTargetLon () override
virtual double GetTargetAlt () override
virtual const WCHAR * GetTargetLatString () override
virtual const WCHAR * GetTargetLonString () override
virtual bool IsEntityTracking () override
virtual unsigned int GetTargetContainerId () override
virtual unsigned int GetSensorMode () override
virtual bool GetCameraLookAtLLA (double &lat, double &lon, double &alt) override
virtual bool IsGlobalRotate () override
virtual void GetGlobalPbh (float &pitch, float &bank, float &heading) override
virtual void ToggleSensorMode () override
virtual void ToggleTracking () override
virtual void TrackWeaponTarget () override
virtual void UntrackWeaponTarget () override
```

Variables

```
REFIID IID_IControllableCameraV450 = __uuidof(IControllableCameraV450)
REFGUID SID_ControllableCamera = __uuidof(IControllableCameraV400)
```

Function Documentation

◆ **ActivateEntityTracking()**

virtual void ActivateEntityTracking ()

pure virtual

Activate Entity Tracking

◆ **ActivatePositionTracking()**

virtual void ActivatePositionTracking ()

pure virtual

Activate Position Tracking

◆ **AddPostProcess()**

virtual void AddPostProcess (const WCHAR * postProcess)

pure virtual

Add post process

Parameters

postProcess name of post process

◆ ClearPostProcess()

virtual void ClearPostProcess ()

pure virtual

Clear post processes

◆ DeactivateEntityTracking()

virtual void DeactivateEntityTracking ()

pure virtual

Deactivate Entity Tracking

◆ DeactivatePositionTracking()

virtual void DeactivatePositionTracking ()

pure virtual

Deactivate Position Tracking

◆ DeInit()

virtual HRESULT DeInit (void)

pure virtual

Deinit the Controllable Camera

◆ GetBank()

virtual float GetBank ()

pure virtual

Get camera bank in degrees

Returns

Camera bank in degrees

◆ GetCameraLookAtLLA()

virtual bool GetCameraLookAtLLA (double & **lat**,
double & **lon**,
double & **alt**
)

pure virtual

Get camera lookat location LLA in degrees/meters

Parameters

- [out] **lat** latitude in degrees
- [out] **lon** longitude in degrees
- [out] **alt** altitude in meters

◆ GetFov()

```
virtual void GetFov ( float & hFov,  
                      float & vFov  
                    )
```

pure virtual

Get camera's horizontal and vertical field of view in degrees

Parameters

- [out] **hFov** Camera's horizontal FOV in degrees
- [out] **vFov** Camera's vertical FOV in degrees

◆ GetGlobalPbh()

```
virtual void GetGlobalPbh ( float & pitch,  
                           float & bank,  
                           float & heading  
                         )
```

pure virtual

Get camera's global pitch bank and heading in degrees

Parameters

- pitch[out]** Camera pitch in degrees
- bank[out]** Camera bank in degrees
- heading[out]** Camera heading in degrees

◆ GetHeading()

```
virtual float GetHeading ( )
```

pure virtual

Get camera heading bank and heading in degrees

Returns

Camera heading in degrees

◆ GetHfov()

```
virtual float GetHfov ( )
```

pure virtual

Get camera's horizontal field of view in degrees

Returns

Camera's horizontal FOV in degrees

◆ **GetPbh()**

```
virtual void GetPbh ( float & pitch,  
                      float & bank,  
                      float & heading  
                  )
```

pure virtual

Get camera pitch bank and heading in degrees

Parameters

pitch[out] Camera pitch in degrees
bank[out] Camera bank in degrees
heading[out] Camera heading in degrees

◆ **GetPitch()**

```
virtual float GetPitch ( )
```

pure virtual

Get camera pitch bank and heading in degrees

Returns

Camera pitch in degrees

◆ **GetSensorMode()**

```
virtual unsigned int GetSensorMode ( )
```

pure virtual

Get sensor mode

Returns

sensor mode

◆ **GetTargetAlt()**

```
virtual double GetTargetAlt ( )
```

pure virtual

Get target altitude in meters

Returns

target altitude in meters

◆ GetTargetContainerId()

virtual unsigned int GetTargetContainerId()

pure virtual

Get target container id

Returns

target container id

◆ GetTargetLat()

virtual double GetTargetLat()

pure virtual

Get target latitude degrees

Returns

target latitude in degrees

◆ GetTargetLatString()

virtual const WCHAR* GetTargetLatString()

pure virtual

Get target latitude as string

Returns

target altitude in meters

◆ GetTargetLla()

virtual void GetTargetLla(double & **lat**,
 double & **lon**,
 double & **alt**
)

pure virtual

Get target latitude longitude and altitude in degrees/meters

Parameters

[out] **lat** Camera latitude in degrees
[out] **lon** Camera longitude in degrees
[out] **alt** Camera altitude in meters

◆ GetTargetLon()

virtual double GetTargetLon()

pure virtual

Get target longitude degrees

Returns

target longitude in degrees

◆ GetTargetLonString()

virtual const WCHAR* GetTargetLonString()

pure virtual

Get target longitude as string

Returns

target longitude in meters

◆ GetVfov()

virtual float GetVfov()

pure virtual

Get camera's vertical field of view in degrees

Returns

Camera's vertical FOV in degrees

◆ GetX()

virtual float GetX()

pure virtual

Get camera x offset in meters

Returns

Camera x offset in meters

◆ GetXyz()

virtual void GetXyz(float & **x**,
 float & **y**,
 float & **z**
)

pure virtual

Get camera xyz offset in meters

Parameters

[out] **x** Camera x offset in meters
[out] **y** Camera y offset in meters
[out] **z** Camera z offset in meters

◆ GetY()

`virtual float GetY()`

`pure virtual`

Get camera y offset in meters

Returns

Camera y offset in meters

◆ `GetZ()`

`virtual float GetZ()`

`pure virtual`

Get camera z offset in meters

Returns

Camera z offset in meters

◆ `GetZoom()`

`virtual float GetZoom()`

`pure virtual`

Get camera zoom

Returns

Camera zoom

◆ `Init()`

`virtual HRESULT Init(LPCWSTR sWindowName)`

`pure virtual`

Initialize the Controllable Camera system with the window name

Parameters

sWindowName Name of window

◆ `IsEntityTracking()`

`virtual bool IsEntityTracking()`

`pure virtual`

Check if entity tracking is enabled

Returns

true if entity tracking is enabled

◆ `IsGlobalRotate()`

`virtual bool IsGlobalRotate()`

`pure virtual`

Check if global rotation is enabled

Returns

true if global rotation is enabled

◆ **IsInitialized()**

virtual bool IsInitialized () const

pure virtual

Returns true if camera is initialized

◆ **IsPositionTracking()**

virtual bool IsPositionTracking ()

pure virtual

Is position tracking enabled

Returns

true if position tracking is enabled

◆ **IsRegistered()**

virtual bool IsRegistered ()

pure virtual

Check if view is registered

◆ **IsVisible()**

virtual bool IsVisible ()

pure virtual

Is view being displayed setting this does not modify the **P3D** camera in any way.

Returns

true if view is visible

◆ **Register()**

virtual HRESULT Register ()

pure virtual

Register view

◆ **RemovePostProcess()**

virtual void RemovePostProcess (const WCHAR * postProcess)

pure virtual

Remove post process

Parameters

postProcess name of post process

◆ **ResetPostProcess()**

virtual void ResetPostProcess()

pure virtual

Reset post processes

◆ **SetBank()**

virtual void SetBank (float **bank**)

pure virtual

Set camera bank in degrees

Parameters

bank Camera bank in degrees

◆ **SetContinuousBank()**

virtual void SetContinuousBank (float **bank**)

pure virtual

Set Continuous bank

Parameters

pitch bank in meters

◆ **SetContinuousHeading()**

virtual void SetContinuousHeading (float **heading**)

pure virtual

Set Continuous Pitch

Parameters

pitch pitch in meters

◆ **SetContinuousPitch()**

virtual void SetContinuousPitch (float **pitch**)

pure virtual

Set Continuous heading

Parameters

pitch heading in meters

◆ **SetFov()**

```
virtual void SetFov ( float hFov,  
                      float vFov  
                    )
```

pure virtual

Set camera's horizontal and vertical field of view in degrees

Parameters

hFov Camera horizontal FOV in degrees

◆ **SetGlobalPbh()**

```
virtual void SetGlobalPbh ( float pitch,  
                           float bank,  
                           float heading  
                         )
```

pure virtual

Set camera pitch bank and heading in degrees with global values

Parameters

pitch Camera pitch in degrees

bank Camera bank in degrees

heading Camera heading in degrees

◆ **SetGlobalRotate()**

```
virtual void SetGlobalRotate ( bool bGlobalRotate )
```

pure virtual

Set global rotate

Parameters

bGlobalRotate enables global rotation

◆ **SetHeading()**

```
virtual void SetHeading ( float heading )
```

pure virtual

Set camera heading in degrees

Parameters

heading Camera heading in degrees

◆ SetHfov()

virtual void SetHfov (float **hFov**)

pure virtual

Set camera's vertical field of view in degrees

Parameters

hFov Camera horizontal FOV in degrees

◆ SetPbh()

virtual void SetPbh (float **pitch**,
 float **bank**,
 float **heading**
)

pure virtual

Set camera pitch bank and heading in degrees

Parameters

pitch Camera pitch in degrees
bank Camera bank in degrees
heading Camera heading in degrees

◆ SetPitch()

virtual void SetPitch (float **pitch**)

pure virtual

Set camera pitch in degrees

Parameters

pitch Camera pitch in degrees

◆ SetSensorMode()

virtual void SetSensorMode (unsigned int **sensorMode**)

pure virtual

Set camera sensor mode

Parameters

sensorMode sensor mode

◆ SetTargetAlt()

virtual void SetTargetAlt (double **alt**)

pure virtual

Set target altitude in degrees/meters

Parameters

alt target altitude in meters

◆ SetTargetContainerId()

virtual void SetTargetContainerId (unsigned int **targetId**)

pure virtual

Set target container id

Parameters

targetID container id of target

◆ SetTargetLat()

virtual void SetTargetLat (double **lat**)

pure virtual

Set target latitude longitude and altitude in degrees/meters

Parameters

lat target latitude in degrees

◆ SetTargetLla()

virtual void SetTargetLla (double **lat**,
 double **lon**,
 double **alt**
)

pure virtual

Set target latitude longitude and altitude in degrees/meters

Parameters

lat target latitude in degrees

lon target longitude in degrees

alt target altitude in meters

◆ SetTargetLon()

virtual void SetTargetLon (double **lon**)

pure virtual

Set target longitude in degrees/meters

Parameters

lon target longitude in degrees

◆ SetVfov()

virtual void SetVfov (float **vFov**)

pure virtual

Set camera's vertical field of view in degrees

Parameters

vFov Camera vertical FOV in degrees

◆ **SetVisible()**

virtual void SetVisible (bool **visible**)

pure virtual

Provides way for external implementations to keep track of when view should be displayed or not, setting this does not modify the **P3D** camera in any way.

Parameters

visible determines if view should be visible or not.

◆ **SetX()**

virtual void SetX (float **x**)

pure virtual

Set camera x offset in meters

Parameters

x Camera x offset in meters

◆ **SetXyz()**

virtual void SetXyz(float **x**,
 float **y**,
 float **z**
)

pure virtual

Set camera xyz offset in meters

Parameters

x Camera x offset in meters
y Camera y offset in meters
z Camera z offset in meters

◆ **SetY()**

virtual void SetY (float **y**)

pure virtual

Set camera y offset in meters

Parameters

y Camera y offset in meters

◆ SetZ()

virtual void SetZ (float **z**)

pure virtual

Set camera z offset in meters

Parameters

z Camera z offset in meters

◆ SetZoom()

virtual void SetZoom (float **zoom**)

pure virtual

Set camera zoom

Parameters

zoom camera zoom

◆ ToggleSensorMode()

virtual void ToggleSensorMode ()

pure virtual

Toggles between all available sensor modes on the camera

◆ ToggleTracking()

virtual void ToggleTracking ()

pure virtual

Toggles tracking on or off. Tries to track an entity first, if none are nearby falls back to tracking the ground location

◆ TrackWeaponTarget()

virtual void TrackWeaponTarget ()

pure virtual

Set the camera to track the current weapon system target (fire control system)

◆ UntrackWeaponTarget()

virtual void UntrackWeaponTarget ()

pure virtual

Stop camera from tracking the current weapon system target (fire control system)

◆ **ZoomIn()**

virtual void ZoomIn()

pure virtual

Zoom camera in

◆ **ZoomOut()**

virtual void ZoomOut()

pure virtual

Zoom camera out

Variable Documentation

◆ **IID_IControllableCameraV450**

REFIID IID_IControllableCameraV450 = __uuidof(**IControllableCameraV450**)

◆ **SID_ControllableCamera**

REFGUID SID_ControllableCamera = __uuidof(IControllableCameraV400)

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Menu Service

Overview

This service provides access to Prepar3D's menu system. It can be used to add custom menu items to the application menus.

Classes

class [IMenuServiceV410](#)

class [MenuItemV410](#)

Class Documentation

◆ P3D::IMenuServiceV410

class P3D::IMenuServiceV410

This service provides access to Prepar3D's menu system.

Inherits IUnknown.

Private Member Functions

virtual [MenuItemV410](#) * [CreateMenuItem](#) () **PURE**

 virtual HRESULT [AddItem](#) (USHORT uMenuID, USHORT uParentID, int nInsertIndex)
 PURE

 virtual HRESULT [RemoveItem](#) (USHORT ultemID, USHORT uParentID) **PURE**

Member Function Documentation

◆ [AddItem\(\)](#)

```
virtual HRESULT AddItem( USHORT uMenuItem,
                        USHORT uParentID,
                        int      nInsertIndex
)                                [private] [virtual]
```

Add menu item to menu. Item will be removed current position if it has already been added to the menu.

Parameters

- uMenuItem** The menu ID to be added.
- uParentID** The parent ID of the menu ID to be added.

Returns

S_OK if the icon instance was successfully added, E_FAIL otherwise.

◆ CreateMenuItem()

```
virtual IMenultemV410* CreateMenuItem( ) [private] [virtual]
```

Creates an menu item instance.

Returns

An menu item instance.

◆ RemoveItem()

```
virtual HRESULT RemoveItem( USHORT ultemID,
                           USHORT uParentID
)                                [private] [virtual]
```

Removes a developer defined menu item.

Parameters

- ultemID** The menu ID to be destroyed.

Returns

S_OK if the icon instance was successfully removed, E_FAIL otherwise.

◆ P3D::IMenultemV410

class P3D::IMenultemV410

This interface may be used by developers to manipulate menu items.

Inherits IUnknown.

Private Member Functions

```
virtual USHORT GetId () const PURE
virtual LPCWSTR GetText () const PURE
virtual void SetText (LPCWSTR text) PURE
virtual MenuTypePdk GetType () const PURE
virtual void SetType (MenuTypePdk type) PURE
virtual bool IsChecked () const PURE
virtual void SetChecked (bool checked) PURE
virtual void SetActivated (bool activated) PURE
virtual void RegisterCallback (ICallbackV400 *pCallback) PURE
virtual void UnregisterCallback (ICallbackV400 *pCallback) PURE
```

Member Function Documentation

◆ **GetId()**

```
virtual USHORT GetId ( ) const private virtual
```

◆ **GetText()**

```
virtual LPCWSTR GetText ( ) const private virtual
```

◆ **GetType()**

```
virtual MenuTypePdk GetType ( ) const private virtual
```

◆ **IsChecked()**

```
virtual bool IsChecked ( ) const private virtual
```

◆ **RegisterCallback()**

```
virtual void RegisterCallback (ICallbackV400 * pCallback) private virtual
```

◆ **SetActivated()**

```
virtual void SetActivated (bool activated) private virtual
```

◆ **SetChecked()**

```
virtual void SetChecked (bool checked) private virtual
```

◆ SetText()

```
virtual void SetText( LPCWSTR text ) [private] [virtual]
```

◆ SetType()

```
virtual void SetType( MenuTypePdk type ) [private] [virtual]
```

◆ UnregisterCallback()

```
virtual void UnregisterCallback( ICallbackV400 * pCallback ) [private] [virtual]
```

Typedefs

```
using IMenulist = IListBuilder<IMenulistV410>  
using Menulist = CComPtrVecBuilder<IMenulistV410>
```

Enumerations

```
enum MenuTypePdk {  
    MENU_SEPARATOR_ITEM = 0, MENU_ITEM, MENU_CHECK_ITEM,  
    MENU_BAR_ITEM,  
    MENU_ROOT  
}
```

Variables

```
GUID IID_IMenuServiceV410  
GUID SID_MenuService  
GUID P3D_EVENT_GUID  
GUID IID_IMenulistV410  
GUID SID_IMenulistV410
```

Typeface Documentation

◆ IMenulist

```
using IMenulist = IListBuilder<IMenulistV410>
```

◆ Menulist

```
using Menulist = CComPtrVecBuilder<IMenulistV410>
```

Enumeration Type Documentation

◆ **MenuTypePdk**

enum **MenuTypePdk**

Enumerator

MENU_SEPARATOR_ITEM
MENU_ITEM
MENU_CHECK_ITEM
MENU_BAR_ITEM
MENU_ROOT

Variable Documentation

◆ **IID_IMenuItemV410**

GUID IID_IMenuItemV410

◆ **IID_IMenuServiceV410**

GUID IID_IMenuServiceV410

◆ **P3D_EVENT_GUID**

GUID P3D_EVENT_GUID

◆ **SID_IMenuItemV410**

GUID SID_IMenuItemV410

◆ **SID_MenuService**

GUID SID_MenuService

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Network Services

Overview

This PDK service allows callers to manipulate various network systems during runtime.

Classes

- class **IMultiplayerServiceV453**
- class **IMultichannelServiceV440**
- class **IComponentControlV430**
- class **IComponentControlCallbackV430**
- class **ICigiPacketV440**
- class **ICigiPacketCallbackV440**
- class **ICigiServiceV430**

Class Documentation

◆ P3D::IMultiplayerServiceV453

class P3D::IMultiplayerServiceV453

This is the interface to the core Prepar3D multiplayer system. An example on how to query for a PDK service can be found in the DLLStart() function of the Camera PDK Sample.

Inherits IMultiplayerServiceV430.

Public Member Functions

- virtual bool **InSession** () const override
- virtual bool **IsHosting** () const override
- virtual bool **IsAntiCheatEnabled** () const override
- virtual bool **IsSlewModeEnabled** () const override
- virtual bool **IsPauseLocalSimulationEnabled** () const override
- virtual uint **GetMaxPlayerCount** () const override
- virtual double **GetTimeSinceStart** () const override
- virtual uint **GetPlayerCount** () const override
- virtual uint **GetPlayerObjectID** (in uint ulIndex) const override
- virtual HRESULT **GetPlayerName** (in uint ulIndex, out LPWSTR pszName, in uint uLength) const override
- virtual HRESULT **GetPlayerRoleGUID** (in uint ulIndex, out GUID &guidRole) const override
- virtual HRESULT **GetAircraftTitle** (in uint ulIndex, out LPWSTR pszTitle, in uint uLength) const override

Member Function Documentation

◆ GetAircraftTitle()

```
virtual HRESULT GetAircraftTitle ( in uint ulIndex,
                                  out LPWSTR pszTitle,
                                  in uint uLength
                                ) const [pure virtual]
```

Gets the aircraft title for the player.

Parameters

- pszTitle** The buffer to copy the player aircraft title into.
- uLength** The length of the buffer to copy the player aircraft title into.

Returns

Returns S_OK if the title was successfully retrieved and copied, E_FAIL otherwise.

◆ GetMaxPlayerCount()

virtual UINT GetMaxPlayerCount() const **pure virtual**

Gets the maximum number of players in for session.

Returns

Returns the maximum number of players for the current session.

◆ GetPlayerCount()

virtual UINT GetPlayerCount() const **pure virtual**

Gets the number of players in the session.

Returns

Returns the number of players in the current session.

◆ GetPlayerName()

```
virtual HRESULT GetPlayerName( __in UINT ulIndex,
                             __out LPWSTR pszName,
                             __in UINT uLength
                           ) const pure virtual
```

Gets the name of the player at the given index.

Parameters

ulIndex The index of the player.
pszName The buffer to copy the player name into.
uLength The length of the buffer to copy the player name into.

Returns

Returns S_OK if the name was successfully retrieved and copied, E_FAIL otherwise.

Remarks

Indexes are 0 based and the max index will be [GetPlayerCount\(\)](#) - 1. Multiplayer role indexes may changed as users join and leave the session.

◆ GetPlayerObjectID()

virtual UINT GetPlayerObjectID(__in UINT ulIndex) const **pure virtual**

Gets the object ID of the player at the given index.

Parameters

ulIndex The index of the player.

Returns

Returns the player's object ID.

Remarks

A return value of 0 is an invalid object ID. Indexes are 0 based and the max index will be [GetPlayerCount\(\)](#) - 1. Multiplayer role indexes may changed as users join and leave the session.

◆ GetPlayerRoleGUID()

```
virtual HRESULT GetPlayerRoleGUID( __in UINT ulIndex,
                                   __out GUID & guidRole
                                 ) const pure virtual
```

Gets the player role GUID of the player at the given index.

Parameters

ulIndex The index of the player.
guidRole The player role GUID.

Returns

Returns S_OK if the multiplayer role GUID for the given player index was successfully found, E_FAIL otherwise.

Remarks

Indexes are 0 based and the max index will be [GetPlayerCount\(\)](#) - 1. Multiplayer role indexes may changed as users join and leave the session. This function only succeeds when in a multiplayer structured scenario.

◆ **GetTimeSinceStart()**

virtual double GetTimeSinceStart() const **pure virtual**

Gets the time since the session started.

Returns

Returns the time since the session started.

◆ **InSession()**

virtual bool InSession() const **pure virtual**

Returns true if the client is currently connected to a multiplayer session.

Remarks

The client may still be in the lobby at this point.

◆ **IsAntiCheatEnabled()**

virtual bool IsAntiCheatEnabled() const **pure virtual**

Gets a value indicating whether or not anti-cheat is enabled.

Returns

Returns a value indicating whether or not anti-cheat is enabled.

◆ **IsHosting()**

virtual bool IsHosting() const **pure virtual**

Returns true if the client is currently connected to a multiplayer session and is the host.

Remarks

The client may still be in the lobby at this point.

◆ **IsPauseLocalSimulationEnabled()**

virtual bool IsPauseLocalSimulationEnabled() const **pure virtual**

Gets a value indicating whether or not the local sims can pause the simulation.

Returns

Returns a value indicating whether or not the local sims can pause the simulation.

◆ **IsSlewModeEnabled()**

virtual bool IsSlewModeEnabled() const **pure virtual**

Gets a value indicating whether or not the local sims can slew.

Returns

Returns a value indicating whether or not the local sims can slew.

◆ **P3D::IMultichannelServiceV440**

class P3D::IMultichannelServiceV440

This is the interface to the core Prepar3D multiplayer system. An example on how to query for a PDK service can be found in the DLLStart() function of the Camera PDK Sample.

Inherits IUnknown.

Public Member Functions

virtual bool **InSession**() const override

virtual bool **IsMaster**() const override

virtual bool **IsSlave**() const override

Member Function Documentation

◆ **InSession()**

`virtual bool InSession() const pure virtual`

Returns true if the client is currently connected to a multichannel session.

Remarks

The client may still be in the lobby at this point.

◆ **IsMaster()**

`virtual bool IsMaster() const pure virtual`

Returns true if the client is the multichannel master

◆ **IsSlave()**

`virtual bool IsSlave() const pure virtual`

Returns true if the client is a multichannel slave

◆ **P3D::IComponentControlV430**

class P3D::IComponentControlV430

This interface is used by a plugin to populate Component Control data when sending and receiving packets. This interface allows for setting/getting of bytes, shorts, integers, floats, doubles, and 64-bit integers.

Remarks

When setting a byte or short, the position of the byte and short must also be given.

When setting a 64-bit value, valid word indexes are [0,2]. See also the CigiComponentControl PDK sample.

Inherits IUnknown.

Private Member Functions

<code>virtual void SendData() override</code>
<code>virtual void ClearData() override</code>
<code>virtual void SetShortComponent(bool blsShort) override</code>
<code>virtual bool GetShortComponent() const override</code> Gets the short component flag. More...
<code>virtual void SetComponentID(UINT16 uID) override</code> Sets the Component ID. More...
<code>virtual UINT16 GetComponentID() const override</code> Gets the Component ID. More...
<code>virtual void SetComponentClass(UINT8 uClass) override</code> Sets the Component Class. More...
<code>virtual UINT8 GetComponentClass() const override</code> Gets the Component Class. More...
<code>virtual void SetComponentState(UINT8 uState) override</code> Sets the Component State. More...
<code>virtual UINT8 GetComponentState() const override</code> Gets the Component State. More...
<code>virtual void SetInstanceID(UINT16 ulnstanceID) override</code> Sets the Instance ID. More...
<code>virtual UINT16 GetInstanceID() const override</code> Gets the Instance ID. More...
<code>virtual void SetUCharData(UINT8 uData, UINT uWord, P3D::BYTE_POS ePos) override</code> Sets an unsigned byte for the given word in the given byte position. More...
<code>virtual UINT8 GetUCharData(UINT uWord, P3D::BYTE_POS ePos) const override</code> Gets an unsigned byte from the given word in the given byte position. More...
<code>virtual void SetCharData(INT8 iData, UINT uWord, P3D::BYTE_POS ePos) override</code> Sets a byte for the given word in the given byte position. More...
<code>virtual INT8 GetCharData(UINT uWord, P3D::BYTE_POS ePos) const override</code> Gets a byte from the given word in the given byte position. More...
<code>virtual void SetUShortData(UINT16 uData, UINT uWord, P3D::SHORT_POS ePos) override</code> Sets a 16-bit unsigned short for the given word in the given short position. More...
<code>virtual UINT16 GetUShortData(UINT uWord, P3D::SHORT_POS ePos) const override</code> Gets a 16-bit unsigned short from the given word in the given short position. More...

```
virtual void SetShortData (INT16 iData, UINT uWord, P3D::SHORT_POS ePos) override
    Sets a 16-bit short for the given word in the given short position. More...

virtual INT16 GetShortData (UINT uWord, P3D::SHORT_POS ePos) const override
    Gets a 16-bit short from the given word in the given short position. More...

virtual void SetUIntData (UINT32 uData, UINT uWord) override
    Sets a 32-bit unsigned integer for the given word. More...

virtual UINT32 GetUIntData (UINT uWord) const override
    Gets a 32-bit unsigned integer from the given word. More...

virtual void SetIntData (INT32 iData, UINT uWord) override
    Sets a 32-bit integer for the given word. More...

virtual INT32 GetIntData (UINT uWord) const override
    Gets a 32-bit integer from the given word. More...

virtual void SetFloatData (float fData, UINT uWord) override
    Sets a 32-bit floating point value for the given word. More...

virtual float GetFloatData (UINT uWord) const override
    Gets a 32-bit floating point value from the given word. More...

virtual void SetUInt64Data (UINT64 uData, UINT uWord) override
virtual UINT64 GetUInt64Data (UINT uWord) const override
virtual void SetDoubleData (double dData, UINT uWord) override
virtual double GetDoubleData (UINT uWord) const override
```

Member Function Documentation

◆ **ClearData()**

```
virtual void ClearData ( ) private pure virtual
```

Resets the IComponentControl data to the default values. Useful for resetting the structure between [SendData\(\)](#) calls, if necessary.

◆ **GetCharData()**

```
virtual INT8 GetCharData ( UINT uWord,
                           P3D::BYTE_POS ePos
                         ) const private pure virtual
```

Gets a byte from the given word in the given byte position.

◆ **GetComponentClass()**

```
virtual UINT8 GetComponentClass ( ) const private pure virtual
```

Gets the Component Class.

◆ **GetComponentID()**

```
virtual UINT16 GetComponentID ( ) const private pure virtual
```

Gets the Component ID.

◆ **GetComponentState()**

```
virtual UINT8 GetComponentState ( ) const private pure virtual
```

Gets the Component State.

◆ **GetDoubleData()**

```
virtual double GetDoubleData ( UINT uWord ) const private pure virtual
```

Gets a 64-bit double floating point value from the given word. Valid word values for 64-bit data fields are [0,2].

◆ **GetFloatData()**

```
virtual float GetFloatData ( UINT uWord ) const private pure virtual
```

Gets a 32-bit floating point value from the given word.

◆ **GetInstanceID()**

```
virtual UINT16 GetInstanceID( ) const private pure virtual
```

Gets the Instance ID.

◆ **GetIntData()**

```
virtual INT32 GetIntData( UINT uWord ) const private pure virtual
```

Gets a 32-bit integer from the given word.

◆ **GetShortComponent()**

```
virtual bool GetShortComponent( ) const private pure virtual
```

Gets the short component flag.

◆ **GetShortData()**

```
virtual INT16 GetShortData( UINT uWord,  
                           P3D::SHORT_POS ePos  
                         ) const private pure virtual
```

Gets a 16-bit short from the given word in the given short position.

◆ **GetUCharData()**

```
virtual UINT8 GetUCharData( UINT uWord,  
                           P3D::BYTE_POS ePos  
                         ) const private pure virtual
```

Gets an unsigned byte from the given word in the given byte position.

◆ **GetUInt64Data()**

```
virtual UINT64 GetUInt64Data( UINT uWord ) const private pure virtual
```

Gets a 64-bit unsigned integer value from the given word. Valid word values for 64-bit data fields are [0,2].

◆ **GetUIntData()**

```
virtual UINT32 GetUIntData( UINT uWord ) const private pure virtual
```

Gets a 32-bit unsigned integer from the given word.

◆ **GetUShortData()**

```
virtual UINT16 GetUShortData( UINT uWord,  
                           P3D::SHORT_POS ePos  
                         ) const private pure virtual
```

Gets a 16-bit unsigned short from the given word in the given short position.

◆ **SendData()**

```
virtual void SendData( ) private pure virtual
```

Signals Prepar3D to queue the current Component Control packet data. This function should be called once per Component Control packet, but may also be called more than once per **IComponentControlCallbackV430::OnSend()** callback to send multiple packets.

◆ **SetCharData()**

```
virtual void SetCharData( INT8 iData,  
                        UINT uWord,  
                        P3D::BYTE_POS ePos
```

)	private pure virtual
Sets a byte for the given word in the given byte position.		
◆ SetComponentClass()		
virtual void SetComponentClass (UINT8 uClass) private pure virtual		
Sets the Component Class.		
◆ SetComponentID()		
virtual void SetComponentID (UINT16 uID) private pure virtual		
Sets the Component ID.		
◆ SetComponentState()		
virtual void SetComponentState (UINT8 uState) private pure virtual		
Sets the Component State.		
◆ SetDoubleData()		
virtual void SetDoubleData (double dData, UINT uWord) private pure virtual		
Sets a 64-bit double floating point value for the given word. Valid word values for 64-bit data fields are [0,2].		
◆ SetFloatData()		
virtual void SetFloatData (float fData, UINT uWord) private pure virtual		
Sets a 32-bit floating point value for the given word.		
◆ SetInstanceID()		
virtual void SetInstanceID (UINT16 ulnstanceID) private pure virtual		
Sets the Instance ID.		
◆ SetIntData()		
virtual void SetIntData (INT32 iData, UINT uWord) private pure virtual		
Sets a 32-bit integer for the given word.		
◆ SetShortComponent()		
virtual void SetShortComponent (bool blsShort) private pure virtual		
Sets the Short Component Control flag. If true, Prepar3D will queue a Short Component Control packet with a call to SendData() .		
◆ SetShortData()		
virtual void SetShortData (INT16 <i>iData,</i> UINT <i>uWord,</i> P3D::SHORT_POS <i>ePos</i>) private pure virtual		
Sets a 16-bit short for the given word in the given short position.		

◆ SetUCharData()

```
virtual void SetUCharData( UINT8 uData,  
                           UINT uWord,  
                           P3D::BYTE_POS ePos  
                         ) private pure virtual
```

Sets an unsigned byte for the given word in the given byte position.

◆ SetUInt64Data()

```
virtual void SetUInt64Data( UINT64 uData,  
                           UINT uWord  
                         ) private pure virtual
```

Sets a 64-bit unsigned integer value for the given word. Valid word values for 64-bit data fields are [0,2].

◆ SetUIntData()

```
virtual void SetUIntData( UINT32 uData,  
                           UINT uWord  
                         ) private pure virtual
```

Sets a 32-bit unsigned integer for the given word.

◆ SetUShortData()

```
virtual void SetUShortData( UINT16 uData,  
                           UINT uWord,  
                           P3D::SHORT_POS ePos  
                         ) private pure virtual
```

Sets a 16-bit unsigned short for the given word in the given short position.

◆ P3D::IComponentControlCallbackV430

```
class P3D::IComponentControlCallbackV430
```

This interface is implemented by a plugin to receive Component Control callbacks. Callbacks can be registered/unregistered using the CIGI PDK Service [ICigiServiceV430::RegisterComponentControlCallback\(\)](#) and [ICigiServiceV430::UnregisterComponentControlCallback\(\)](#) functions.

Inherits IUnknown.

Private Member Functions

```
virtual HRESULT OnSend (IComponentControlV430 &CompCtrl) override  
virtual HRESULT OnReceive (const IComponentControlV430 &CompCtrl) override
```

Member Function Documentation

◆ OnReceive()

```
virtual HRESULT OnReceive ( const IComponentControlV430 & CompCtrl ) private pure virtual
```

This function is called once per frame while Prepar3D is an active CIGI IG. Plugins should implement this function and determine if they are to process the received packet.

◆ OnSend()

```
virtual HRESULT OnSend ( IComponentControlV430 & CompCtrl ) private pure virtual
```

This function is called once per frame while Prepar3D is an active CIGI host. Plugins should fill the IComponentControl interface with the necessary Component Control packet data. The plugin should then call [IComponentControlV430::SendData\(\)](#) to signal to Prepar3D that the current data should be queued as an outgoing message. Plugins are able to call SendData() multiple times during this callback to send more than one Component Control packet per frame. The [IComponentControlV430::ClearData\(\)](#) function may be called to reset the packet data between calls to SendData(). The ClearData() function is there mainly to ease development and is not required to be called.

◆ P3D::ICigiPacketV440

class P3D::ICigiPacketV440

Inherits IUnknown.

Private Member Functions

```
virtual USHORT GetPacketID () const override
virtual USHORT GetPacketSize () const override
virtual BYTE GetMajorVersion () const override
virtual BYTE GetMinorVersion () const override
virtual void SetPacketID (USHORT PacketID) override
virtual HRESULT GetData (void *pData, UINT cbSize) const override
virtual HRESULT SetData (void *pData, UINT cbSize) override
virtual HRESULT SendData () override
```

Member Function Documentation

◆ GetData()

```
virtual HRESULT GetData ( void * pData,
                        UINT cbSize
                      ) const [private] [pure virtual]
```

◆ GetMajorVersion()

```
virtual BYTE GetMajorVersion ( ) const [private] [pure virtual]
```

◆ GetMinorVersion()

```
virtual BYTE GetMinorVersion ( ) const [private] [pure virtual]
```

◆ GetPacketID()

```
virtual USHORT GetPacketID ( ) const [private] [pure virtual]
```

◆ GetPacketSize()

```
virtual USHORT GetPacketSize ( ) const [private] [pure virtual]
```

◆ SendData()

```
virtual HRESULT SendData ( ) [private] [pure virtual]
```

◆ SetData()

```
virtual HRESULT SetData ( void * pData,
                        UINT cbSize
                      ) [private] [pure virtual]
```

◆ SetPacketID()

```
virtual void SetPacketID ( USHORT PacketID ) [private] [pure virtual]
```

◆ P3D::ICigiPacketCallbackV440

class P3D::ICigiPacketCallbackV440

Inherits IUnknown.

Private Member Functions

```
virtual HRESULT OnCustomSend (const USHORT &uPacketID, ICigiPacketV440 &pPacket)
                             override
virtual HRESULT OnP3DReceive (const USHORT &uPacketID, ICigiPacketV440 &pPacket)
                            override
```

OVERVIEW

```
virtual HRESULT OnP3DSend (const USHORT &uPacketID, ICigiPacketV440 &pPacket)
override
```

Member Function Documentation

- ◆ **OnCustomSend()**

```
virtual HRESULT OnCustomSend ( const USHORT & uPacketID,
ICigiPacketV440 & pPacket
)
private pure virtual
```

- ◆ **OnP3DReceive()**

```
virtual HRESULT OnP3DReceive ( const USHORT & uPacketID,
ICigiPacketV440 & pPacket
)
private pure virtual
```

- ◆ **OnP3DSend()**

```
virtual HRESULT OnP3DSend ( const USHORT & uPacketID,
ICigiPacketV440 & pPacket
)
private pure virtual
```

◆ **P3D::ICigiServiceV430**

This is the interface to the core Prepar3D CIGI plugin. An example on how to query for a PDK service can be found in the DLLStart() function of the Camera PDK Sample.

Inherits IUnknown.

Public Member Functions

virtual bool InSession () const override
virtual bool IsHosting () const override
virtual bool IsIG () const override
virtual UINT GetMajorVersion () const override
virtual UINT GetMinorVersion () const override
virtual HRESULT GetEntityID (_in UINT uObjectID, _out USHORT &usEntityID) const override
virtual HRESULT GetObjectID (_in USHORT usEntityID, _out UINT &uObjectID) const override
virtual HRESULT RegisterComponentControlCallback (_in __notnull P3D::IComponentControlCallbackV430 *pCallback) override
virtual HRESULT UnregisterComponentControlCallback (_in __notnull P3D::IComponentControlCallbackV430 *pCallback) override
virtual HRESULT RegisterCigiPacketCallback (_in const USHORT &uPacketID, _in __notnull P3D::ICigiPacketCallbackV440 *pCallback) override
virtual HRESULT UnregisterCigiPacketCallback (_in const USHORT &uPacketID, _in __notnull P3D::ICigiPacketCallbackV440 *pCallback) override
virtual HRESULT AddExcludedSendID (_in const USHORT &uCustomPacketID) override
virtual HRESULT RemoveExcludedSendID (_in const USHORT &uCustomPacketID) override
virtual HRESULT AddExcludedReceiveID (_in const USHORT &uCustomPacketID) override
virtual HRESULT RemoveExcludedReceiveID (_in const USHORT &uCustomPacketID) override

Member Function Documentation

- ◆ **AddExcludedReceiveID()**

```
virtual HRESULT AddExcludedReceiveID (_in const USHORT & uCustomPacketID ) pure virtual
```

- ◆ **AddExcludedSendID()**

```
virtual HRESULT AddExcludedSendID (_in const USHORT & uCustomPacketID ) pure virtual
```

- ◆ **GetEntityID()**

```
virtual HRESULT GetEntityID (_in UINT uObjectID,
__out USHORT & usEntityID
)
const pure virtual
```

Gets the CIGI Entity ID for the given Prepar3D Object ID.

Parameters
uObjectID The Prepar3D Object ID.
usEntityID The CIGI Entity ID for the given Prepar3D Object ID.
Returns
Returns S_OK if the CIGI Entity ID for the given Prepar3D Object ID was found, E_FAIL otherwise.
◆ GetMajorVersion()
virtual UINT GetMajorVersion () const pure virtual
Returns the CIGI major version for the current session.
◆ GetMinorVersion()
virtual UINT GetMinorVersion () const pure virtual
Returns the CIGI minor version for the current session.
◆ GetObjectID()
virtual HRESULT GetObjectID (__in USHORT usEntityID, __out uint & uObjectID) const pure virtual
Gets the Prepar3D Object ID for the given CIGI Entity ID.
Parameters
usEntityID The CIGI Entity ID.
uObjectID The Prepar3D Object ID for the given CIGI Entity ID.
Returns
Returns S_OK if the Prepar3D Object ID for the given CIGI Entity ID was found, E_FAIL otherwise.
◆ InSession()
virtual bool InSession () const pure virtual
Returns true if the application is currently connected to a CIGI session.
◆ IsHosting()
virtual bool IsHosting () const pure virtual
Returns true if the application is currently connected to a CIGI session and is the host.
◆ IsIG()
virtual bool IsIG () const pure virtual
Returns true if the application is currently connected to a CIGI session and is the IG.
◆ RegisterCigiPacketCallback()
virtual HRESULT RegisterCigiPacketCallback (__in const USHORT & __in __notnull P3D::ICigiPacketCallbackV440 * pCallback) uPacketID, pCallback pure virtual
Registers the given ICigiPacketCallback.
Parameters
pCallback The callback to be registered.
Returns
Returns S_OK if the callback was successfully registered, E_FAIL otherwise.
Remarks
Plugins will only receive callbacks when in an active CIGI session.
◆ RegisterComponentControlCallback()
virtual HRESULT RegisterComponentControlCallback (__in __notnull P3D::IComponentControlCallbackV430 * pCallback) pure virtual

Registers the given IComponentControlCallback.

Parameters

pCallback The callback to be registered.

Returns

Returns S_OK if the callback was successfully registered, E_FAIL otherwise.

Remarks

Plugins will only receive callbacks when in an active CIGI session.

◆ RemoveExcludedReceiveID()

virtual HRESULT RemoveExcludedReceiveID (__in const USHORT & uCustomPacketID) **pure virtual**

◆ RemoveExcludedSendID()

virtual HRESULT RemoveExcludedSendID (__in const USHORT & uCustomPacketID) **pure virtual**

◆ UnregisterCigiPacketCallback()

virtual HRESULT
UnregisterCigiPacketCallback (__in const USHORT &
__in __notnull P3D::ICigiPacketCallbackV440 * pCallback
)
pure virtual

Unregisters the given ICigiPacketCallback.

Parameters

pCallback The callback to be unregistered.

Returns

Returns S_OK if the callback was successfully unregistered, E_FAIL otherwise.

◆ UnregisterComponentControlCallback()

virtual HRESULT
UnregisterComponentControlCallback (__in __notnull P3D::IComponentControlCallbackV430 * pCallback) **pure virtual**

Unregisters the given IComponentControlCallback.

Parameters

pCallback The callback to be unregistered.

Returns

Returns S_OK if the callback was successfully unregistered, E_FAIL otherwise.

Enumerations

```
enum BYTE_POS { BYTE_POS_3 = 3, BYTE_POS_2 = 2, BYTE_POS_1 = 1, BYTE_POS_0 = 0 }
```

```
enum SHORT_POS { SHORT_POS_1 = 1, SHORT_POS_0 = 0 }
```

Enumeration Type Documentation

◆ BYTE_POS

enum BYTE_POS

The byte position in a component data word where BYTE_POS_0 is the least significant byte and BYTE_POS_3 is the most significant byte.

Enumerator

BYTE_POS_3	Most Significant Byte.
BYTE_POS_2	
BYTE_POS_1	
BYTE_POS_0	Least Significant Byte.

◆ SHORT_POS

enum SHORT_POS

The short position in a component data word where SHORT_POS_1 is the most significant short and SHORT_POS_0 is the least significant short.

Enumerator	
SHORT_POS_1	Most Significant Short.
SHORT_POS_0	Least Significant Short.

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Reporting Service

Overview

This service is provided through the PDK interface and allows callers to report information to the end user via content error logging and alerts.

Namespaces

P3D

Prepar3D SDK namespace used primarily for the PDK and its services.

Classes

class [IReportingServiceV400](#)

Class Documentation

◆ [P3D::IReportingServiceV400](#)

class P3D::IReportingServiceV400

Service used to report information to the user

Inherits IUnknown.

Private Member Functions

```
virtual void ErrorReport (const char *message) override
virtual void ErrorReport (const WCHAR *message) override
virtual AlertWindowButton ShowAlert (const char *message, AlertWindowType
type=Alert_Ok) override
virtual AlertWindowButton ShowAlert (const WCHAR *message, AlertWindowType
type=Alert_Ok) override
```

Member Function Documentation

◆ ErrorReport() [1/2]

virtual void ErrorReport (const char * message) private pure virtual

Send message to content error reporting system

Parameters

message message to log

◆ ErrorReport() [2/2]

virtual void ErrorReport (const WCHAR * message) private pure virtual

Send message to content error reporting system

Parameters

message message to log

◆ ShowAlert() [1/2]

virtual **AlertWindowButton**
 ShowAlert (const char * message,
 AlertWindowType type = **Alert_Ok**
)

private pure virtual

Display an alert box to user with a message

Parameters

message message to log

message window type that determines which buttons will display

Returns

button clicked

◆ ShowAlert() [2/2]

virtual **AlertWindowButton**
 ShowAlert (const WCHAR * message,
 AlertWindowType type = **Alert_Ok**
)

private pure virtual

Display an alert box to user with a message

Parameters

message message to log

message window type that determines which buttons will display

Returns

button clicked

- top -

Related Links

- Data Load Helper
- Visual Effects Service
- Event Service
- Global Data Service
- Rendering Services
- Window and Camera Services
- Weather System Service
- ISimObjects
- PDK Services
- PDK Types
- SDK Overview
- PDK Overview

Sim Property Service

Overview

This service provides the functions for manipulating data stored in SimProp sets

Classes

class **ISimPropertySetFactoryV440**

class **ISimPropertyServiceV440**

Class Documentation

◆ P3D::ISimPropertySetFactoryV440

class P3D::ISimPropertySetFactoryV440

Factory interface implemented by the developer for creating objects from SimProp data

Inherits IUnknown.

Private Member Functions

virtual HRESULT **Create (ISimPropertySet *pSet, IUnknown **pObject)** override

virtual HRESULT **Init (ISimPropertySet *pSet, IUnknown *pObject)** override

virtual HRESULT **Serialize (IUnknown *pObject, ISimPropertySet **pSet)** override

Member Function Documentation

◆ Create()

virtual HRESULT Create (ISimPropertySet * pSet,

```
IUnknown ** pObject  
)  
private pure virtual
```

Callback to create an object from an ISimPropertySet

Parameters

pSet Set from which to create an object
pObject Pointer that will hold the new object

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ Init()

```
virtual HRESULT Init ( ISimPropertySet * pSet,  
IUnknown * pObject  
)  
private pure virtual
```

Callback to initialize an object from an ISimPropertySet

Parameters

pSet Set from which to initialize an object
pObject Object to initialize

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ Serialize()

```
virtual HRESULT Serialize ( IUnknown * pObject,  
ISimPropertySet ** pSet  
)  
private pure virtual
```

Callback to serialize an object into an ISimPropertySet

Parameters

pObject Object to be serialized
pSet Set of properties to serialize into

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ P3D::ISimPropertyServiceV440

class P3D::ISimPropertyServiceV440

This service provides access to data stored in SimProp XML files

Inherits IUnknown.

Private Member Functions

virtual HRESULT	RegisterFactory (const GUID &setGUID, ISimPropertySetFactoryV440 *pFactory) override
virtual HRESULT	UnRegisterFactory (const GUID &setGUID) override
virtual HRESULT	SimPropSetItemRead (IN ISimPropertySet *pSimPropSet, IN OUT PVOID pBaseAddress, IN const SIMPROPSET_ITEM *pPropSetItem)
virtual HRESULT	SimPropSetItemsRead (IN ISimPropertySet *pSimPropSet, IN OUT PVOID pBaseAddress, IN const SIMPROPSET_ITEM aPropSetItems[], IN UINT uCountItems)
virtual HRESULT	SimPropSetItemWrite (IN OUT ISimPropertySet *pSimPropSet, IN PCVOID pBaseAddress, IN const SIMPROPSET_ITEM *pPropSetItem)
virtual HRESULT	SimPropSetItemsWrite (IN OUT ISimPropertySet *pSimPropSet, IN PCVOID pBaseAddress, IN const SIMPROPSET_ITEM pPropSetItems[], IN UINT uCountItems)

Member Function Documentation

◆ RegisterFactory()

```
virtual HRESULT RegisterFactory( const GUID & setGUID,  
                                ISimPropertySetFactoryV440 * pFactory )  
private pure virtual
```

Register a factory that can create objects from an ISimPropertySet

Parameters

setGUID The GUID of the property set
pFactory The factory to register

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ SimPropSetItemRead()

```
virtual HRESULT SimPropSetItemRead( IN ISimPropertySet * pSimPropSet,  
                                    IN OUT PVOID pBaseAddress,  
                                    IN const SIMPROPSET_ITEM * pPropSetItem )  
private virtual
```

Read an item out of a ISimPropertySet into a specific memory location.

This allows property values to be set directly to members of a class or structure.

Parameters

pSimPropSet The property set to read from
pBaseAddress Base memory address to write data
pPropSetItem The property set item description

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ SimPropSetItemsRead()

```
virtual HRESULT
SimPropSetItemsRead( IN ISimPropertySet * pSimPropSet,
IN OUT PVOID pBaseAddress,
IN const SIMPROPSET_ITEM aPropSetItems[],
IN UINT uCountItems
)
```

private virtual

Read multiple items out of a ISimPropertySet into a specific memory location.

This allows property values to be set directly to members of a class or structure.

Parameters

pSimPropSet The property set to read from
pBaseAddress Base memory address to write data
aPropSetItems Array of property set item descriptions
uCountItems Count of property set item descriptions

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ SimPropSetItemsWrite()

```
virtual HRESULT
SimPropSetItemsWrite( IN OUT ISimPropertySet * pSimPropSet,
IN PCVOID pBaseAddress,
IN const SIMPROPSET_ITEM pPropSetItems[],
IN UINT uCountItems
)
```

private virtual

Write multiple items from a specific memory location into an ISimPropertySet. This allows property values to be set directly from members of a class or structure.

Parameters

pSimPropSet The property set to write to
pBaseAddress Base memory address to read from
aPropSetItems Array of property set item descriptions
uCountItems Count of property set item descriptions

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ SimPropSetItemWrite()

```
virtual HRESULT  
SimPropSetItemWrite( [IN OUT] ISimPropertySet * pSimPropSet,  
                     [IN PCVOID] pBaseAddress,  
                     [IN const SIMPROPSET_ITEM * pPropSetItem  
                     )
```

private virtual

Write an item from a specific memory location into an ISimPropertySet. This allows property values to be set directly from members of a class or structure.

Parameters

pSimPropSet The property set to write to
pBaseAddress Base memory address to read from
pPropSetItem The property set item description

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

◆ UnRegisterFactory()

```
virtual HRESULT UnRegisterFactory( const GUID & setGUID ) [private] [pure virtual]
```

Unregister a factory that can create objects from an ISimPropertySet

Parameters

setGUID The GUID of the property set

Returns

HRESULT, S_OK if function succeeds or E_FAIL if it fails

Variables

REFIID IID_ISimPropertyServiceV440 = __uuidof(ISimPropertyServiceV440)

REFIID IID_ISimPropertySetFactoryV440 = __uuidof(ISimPropertySetFactoryV440)

REFIID SID_SimPropertyService = IID_ISimPropertyServiceV440

Variable Documentation

◆ IID_ISimPropertyServiceV440

REFIID IID_ISimPropertyServiceV440 = __uuidof(ISimPropertyServiceV440)

◆ IID_ISimPropertySetFactoryV440

REFIID IID_ISimPropertySetFactoryV440 = __uuidof(ISimPropertySetFactoryV440)

◆ SID_SimPropertyService

REFIID SID_SimPropertyService = **IID_ISimPropertyServiceV440**

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Sound Service

Overview

This PDK service allows callers to manipulate sounds at runtime.

Classes

class **ISoundInstanceV440**

class **ISoundServiceV440**

Class Documentation

◆ P3D::ISoundInstanceV440

class P3D::ISoundInstanceV440

This interface represents a single sound instance.

Inherits IUnknown.

Private Member Functions

virtual HRESULT **IsPlaying** () override

virtual HRESULT **Play** () override

virtual HRESULT **Stop** () override

virtual P3D::SOUND_GROUP **GetSoundGroup** () const override

virtual P3D::VIEW_POINT **GetViewPoint** () const override

virtual HRESULT **SetViewPoint** (__in P3D::VIEW_POINT eViewPoint) override

virtual bool **IsLooping** () const override

virtual bool **Is3D** () const override

virtual bool **IsPitchControlled** () const override

virtual double GetVolume () const override	
virtual HRESULT	SetVolume (<u>in</u> double dVolume) override
virtual double	GetMinVolume () const override
virtual HRESULT	SetMinVolume (<u>in</u> double dMinVolume) override
virtual double	GetMaxVolume () const override
virtual HRESULT	SetMaxVolume (<u>in</u> double dMaxVolume) override
virtual float	GetPitch () const override
virtual HRESULT	SetPitch (<u>in</u> float fPitch) override
virtual HRESULT	GetPosition (<u>out</u> P3D::DXYZ &dxyzLonAltLat) const override
virtual HRESULT	SetPosition (<u>in</u> const P3D::DXYZ &dxyzLonAltLat) override
virtual HRESULT	GetOrientation (<u>out</u> P3D::DXYZ &dxyzOrientation) const override
virtual HRESULT	SetOrientation (<u>in</u> const P3D::DXYZ &dxyzOrientation) override
virtual HRESULT	GetOffset (<u>out</u> P3D::DXYZ &dxyzOffset) const override
virtual HRESULT	SetOffset (<u>in</u> const P3D::DXYZ &dxyzOffset) override
virtual HRESULT	GetVelocity (<u>out</u> P3D::DXYZ &dxyzVelocity) const override
virtual HRESULT	SetVelocity (<u>in</u> const P3D::DXYZ &dxyzVelocity) override
virtual float	GetConePitch () const override
virtual float	GetConeHeading () const override
virtual HRESULT	SetConePitchHeading (<u>in</u> float fConePitch, <u>in</u> float fConeHeading) override
virtual unsigned short	GetConeInsideAngle () const override
virtual unsigned short	GetConeOutsideAngle () const override
virtual HRESULT	SetConeAngles (<u>in</u> unsigned short usInsideConeAngle, <u>in</u> unsigned short usOutsideConeAngle) override
virtual double	GetOutsideConeVolume () const override
virtual HRESULT	SetOutsideConeVolume (<u>in</u> double dOutsideConeVolume) override
virtual float	GetFullScaleDistance () const override
virtual HRESULT	SetFullScaleDistance (<u>in</u> float fFullScaleDistance) override
virtual double	GetCockpitAttenuation () const override
virtual HRESULT	SetCockpitAttenuation (<u>in</u> double dCockpitAttenuation) override

Member Function Documentation

◆ **GetCockpitAttenuation()**

virtual double GetCockpitAttenuation () const **private** **pure virtual**

Returns the cockpit attenuation value. This value is added to the current volume when inside the cockpit.

Remarks

Negative values should be used to decrease volume when in the cockpit, positive values should be used to increase volume.

Units are in 1/100 dBs and values range from -10000 to 10000.

Only applies to 3D sound instances.

◆ GetConeHeading()

virtual float GetConeHeading() const **private** **pure virtual**

Returns the body relative heading of the cone in degrees.

Remarks

Only applies to 3D sound instances.

◆ GetConeInsideAngle()

virtual unsigned short GetConeInsideAngle() const **private** **pure virtual**

Returns the inside angle of the cone in degrees.

Remarks

The default value is 360 and values range from 0 to 360 inclusively.

Only applies to 3D sound instances.

◆ GetConeOutsideAngle()

virtual unsigned short GetConeOutsideAngle() const **private** **pure virtual**

Returns the outside angle of the cone in degrees.

Remarks

The default value is 360 and values range from 0 to 360 inclusively.

Only applies to 3D sound instances.

◆ GetConePitch()

virtual float GetConePitch() const **private** **pure virtual**

Returns the body relative pitch of the cone in degrees.

Remarks

Only applies to 3D sound instances.

◆ GetFullScaleDistance()

virtual float GetFullScaleDistance() const **private** **pure virtual**

Returns the distance in feet where the sound instance begins to attenuate.

Remarks

Only applies to 3D sound instances.

◆ GetMaxVolume()

virtual double GetMaxVolume() const **private** **pure virtual**

Returns the maximum volume that this sound instance will be played.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

◆ GetMinVolume()

virtual double GetMinVolume() const **private** **pure virtual**

Returns the minimum volume that this sound instance will be played.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

◆ GetOffset()

virtual HRESULT GetOffset(__out P3D::DXYZ & dxyzOffset) const **private** **pure virtual**

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzOffset The body relative offset in feet.

Remarks

Only applies to 3D sound instances.

◆ GetOrientation()

virtual HRESULT GetOrientation(__out P3D::DXYZ & dxyzOrientation) const **private** **pure virtual**

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzOrientation The orientation in radians.

Remarks

Only applies to 3D sound instances.

◆ GetOutsideConeVolume()

```
virtual double GetOutsideConeVolume( ) const [private] [pure virtual]
```

Sets the volume of the sound instance when outside of the cone.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

Only applies to 3D sound instances.

◆ GetPitch()

```
virtual float GetPitch( ) const [private] [pure virtual]
```

Returns the current pitch of the sound instance.

Remarks

The default value is 1.0 and values must be greater than 0.0. Pitch control must be enabled when creating the sound instance.

◆ GetPosition()

```
virtual HRESULT GetPosition( __out P3D::DXYZ & dxyzLonAltLat ) const [private] [pure virtual]
```

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzLonAltLat The position in radians/feet.

Remarks

Only applies to 3D sound instances.

◆ GetSoundGroup()

```
virtual P3D::SOUND_GROUP GetSoundGroup( ) const [private] [pure virtual]
```

Returns the **P3D::SOUND_GROUP** of the sound instance.

◆ GetVelocity()

```
virtual HRESULT GetVelocity( __out P3D::DXYZ & dxyzVelocity ) const [private] [pure virtual]
```

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzVelocity The world velocity in feet per second.

Remarks

Only applies to 3D sound instances.

◆ GetViewPoint()

virtual **P3D::VIEW_POINT** GetViewPoint () const **private** **pure virtual**

Returns the **P3D::VIEW_POINT** of the sound instance.

Remarks

Only applies to 3D sound instances.

◆ GetVolume()

virtual double GetVolume () const **private** **pure virtual**

Returns the current volume of the sound instance.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

◆ Is3D()

virtual bool Is3D () const **private** **pure virtual**

Returns true if this is a 3D sound instance, false otherwise.

◆ IsLooping()

virtual bool IsLooping () const **private** **pure virtual**

Returns true if this is a looping sound instance, false otherwise.

◆ IsPitchControlled()

virtual bool IsPitchControlled () const **private** **pure virtual**

Returns true if this sound instances supports pitch control, false otherwise.

◆ IsPlaying()

virtual HRESULT IsPlaying () **private** **pure virtual**

Retruns S_OK if the sound is playing, S_FALSE if it is not, and E_FAIL upon failure.

◆ Play()

virtual HRESULT Play () **private** **pure virtual**

Starts to play the sound instance. Returns S_OK upon success, E_FAIL upon failure.

◆ SetCockpitAttenuation()

virtual HRESULT
SetCockpitAttenuation (__in double dCockpitAttenuation) **private** **pure virtual**

Sets the cockpit attenuation value. This value is added to the current volume when inside the cockpit.

Remarks

Negative values should be used to decrease volume when in the cockpit, positive values should be used to increase volume.

Units are in 1/100 dBs and values range from -10000 to 10000.

Only applies to 3D sound instances.

◆ SetConeAngles()

virtual HRESULT
SetConeAngles (__in unsigned short usInsideConeAngle,
__in unsigned short usOutsideConeAngle
) **private** **pure virtual**

Sets the inside and outside angles of the cone in degrees.

Remarks

The default value is 360 and values range from 0 to 360 inclusively.

Only applies to 3D sound instances.

◆ SetConePitchHeading()

virtual HRESULT SetConePitchHeading (__in float fConePitch,
__in float fConeHeading
) **private** **pure virtual**

Sets the body relative pitch and heading of the cone in degrees.

Parameters

fConePitch The body relative pitch of the cone in degrees.

fConeHeading The body relative heading of the cone in degrees.

Remarks

Only applies to 3D sound instances.

◆ SetFullScaleDistance()

virtual HRESULT SetFullScaleDistance (__in float **fFullScaleDistance**) **private** **pure virtual**

Sets the distance in feet where the sound instance begins to attenuate.

Parameters

fFullScaleDistance The full scale distance in feet.

Remarks

Only applies to 3D sound instances.

◆ SetMaxVolume()

virtual HRESULT SetMaxVolume (__in double **dMaxVolume**) **private** **pure virtual**

Sets the maximum volume that this sound instance will be played.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

◆ SetMinVolume()

virtual HRESULT SetMinVolume (__in double **dMinVolume**) **private** **pure virtual**

Sets the minimum volume that this sound instance will be played.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

◆ SetOffset()

virtual HRESULT SetOffset (__in const P3D::DXYZ & **dxyzOffset**) **private** **pure virtual**

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzOffset The body relative offset in feet.

Remarks

Only applies to 3D sound instances.

◆ SetOrientation()

virtual HRESULT

SetOrientation

(__in const P3D::DXYZ & dxyzOrientation) private pure virtual

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzOrientation The position in radians.

Remarks

Only applies to 3D sound instances.

◆ SetOutsideConeVolume()

virtual HRESULT

SetOutsideConeVolume

(__in double dOutsideConeVolume) private pure virtual

Returns the volume of the sound instance when outside of the cone.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

Only applies to 3D sound instances.

◆ SetPitch()

virtual HRESULT SetPitch (__in float fPitch) private pure virtual

Sets the current pitch of the sound instance.

Remarks

The default value is 1.0 and values must be greater than 0.0. Pitch control must be enabled when creating the sound instance.

◆ SetPosition()

virtual HRESULT SetPosition (__in const P3D::DXYZ & dxyzLonAltLat) private pure virtual

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzLonAltLat The position in radians/feet.

Remarks

Only applies to 3D sound instances.

◆ SetVelocity()

virtual HRESULT SetVelocity (__in const P3D::DXYZ & dxyzVelocity) [private] [pure virtual]

Returns S_OK upon success, E_FAIL otherwise.

Parameters

dxyzVelocity The world velocity in feet per second.

Remarks

Only applies to 3D sound instances.

◆ SetViewPoint()

virtual HRESULT SetViewPoint (__in P3D::VIEW_POINT eViewPoint) [private] [pure virtual]

Sets the **P3D::VIEW_POINT** of the sound instance.

Remarks

Only applies to 3D sound instances.

◆ SetVolume()

virtual HRESULT SetVolume (__in double dVolume) [private] [pure virtual]

Sets the current volume of the sound instance.

Remarks

Units are in 1/100 dBs and values range from 0 to 10000.

◆ Stop()

virtual HRESULT Stop () [private] [pure virtual]

Stops the sound instance. Returns S_OK upon success, E_FAIL upon failure.

◆ P3D::ISoundServiceV440

class P3D::ISoundServiceV440

This PDK service provides developers with the ability to create and manipulate sounds at runtime.

Inherits IUnknown.

Private Member Functions

virtual HRESULT **CreateSoundInstance** (__in LPCWSTR pszFilename, __in
P3D::SOUND_GROUP eSoundGroup, __in P3D::VIEW_POINT
eViewPoint, __in bool bLooping, __in bool b3D, __in bool bPitchControl, __out
void **ppSoundInstance) override

void `ppSoundInstance`; override

Member Function Documentation

◆ CreateSoundInstance()

```
virtual HRESULT  
CreateSoundInstance( __in LPCWSTR pszFilename,  
                     __in P3D::SOUND_GROUP eSoundGroup,  
                     __in P3D::VIEW_POINT eViewPoint,  
                     __in bool bLooping,  
                     __in bool b3D,  
                     __in bool bPitchControl,  
                     __out void ** ppSoundInstance  
)
```

private pure virtual

Creates a reference counted ISoundInstance object. Returns S_OK upon success, E_FAIL otherwise.

Parameters

pszFilename	The filename of the sound file (.wav or .ogg format).
eSoundGroup	The P3D::SOUND_GROUP that the sound instance is associated with.
eViewPoint	The P3D::VIEW_POINT that the sound instance is associated with.
bLooping	True if the sound instance should loop, false otherwise.
b3D	True if the sound instance should support 3D functionality such as position and velocity, false otherwise.
bPitchControl	True if the sound instance should support pitch control, false otherwise. This is only necessary if using the SetPitch function.
ppSoundInstance	The newly created ISoundInstance with reference count 1. QueryInterface should be called on this object to determine the current version of the returned interface. This object should be released when it is no longer needed (either via smart pointer or explicit call to Release()).

Variables

GUID **IID_ISoundInstanceV440**
SoundInstance interface ID. [More...](#)

GUID **IID_ISoundServiceV440**
SoundService interface ID. [More...](#)

GUID **SID_SoundService**
SoundService service ID. [More...](#)

Variable Documentation

◆ IID_ISoundInstanceV440

GUID IID_ISoundInstanceV440

SoundInstance interface ID.

◆ IID_ISoundServiceV440

GUID IID_ISoundServiceV440

SoundService interface ID.

◆ SID_SoundService

GUID SID_SoundService

SoundService service ID.

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

Virtual Reality Service

Overview

This Service allows the caller to interact with the Virtual Reality Interface, tracked controllers, access Virtual Reality Settings and register for specific VR callbacks. It also gives the ability to Enable/Disable VR.

Classes

- struct [VRControllerButton](#)
 - struct [VRControllerAxis](#)
 - struct [P3DEyeData](#)
 - class [IVRTrackedControllerV450](#)
 - class [IVRInterfaceV451](#)
 - class [IVRSettingsV500](#)
 - class [IVRServiceV452](#)
-

Class Documentation

◆ P3D::VRControllerButton

struct P3D::VRControllerButton

Public Member Functions

- bool [WasPressed \(\)](#)
- bool [WasReleased \(\)](#)

Public Attributes

enum [VRControllerButtonType](#) [m_eButtonType](#)

bool **m_bPressedState**
bool **m_bPreviousPressedState**

Member Function Documentation

◆ WasPressed()

bool WasPressed () **inline**

◆ WasReleased()

bool WasReleased () **inline**

Member Data Documentation

◆ m_bPressedState

bool m_bPressedState

◆ m_bPreviousPressedState

bool m_bPreviousPressedState

◆ m_eButtonType

enum **VRControllerButtonType** m_eButtonType

◆ P3D::VRControllerAxis

struct P3D::VRControllerAxis

Public Member Functions

bool **WasPressed ()**
bool **WasReleased ()**

Public Attributes

enum **VRControllerAxisType** m_eAxisType
float **m_fAxisX**
float **m_fAxisY**
bool **m_bPressedState**
bool **m_bPreviousPressedState**

Member Function Documentation

◆ WasPressed()

bool WasPressed () inline

◆ WasReleased()

bool WasReleased () inline

Member Data Documentation

◆ m_bPressedState

bool m_bPressedState

◆ m_bPreviousPressedState

bool m_bPreviousPressedState

◆ m_eAxisType

enum VRControllerAxisType m_eAxisType

◆ m_fAxisX

float m_fAxisX

◆ m_fAxisY

float m_fAxisY

◆ P3D::P3DEyeData

struct P3D::P3DEyeData

Class Members

ARGBColor

DisplayColor

XYZMeters

fGazeDirection

float

fOpenness

float	fPupilDiameter
XYZMeters	LocalFocus
ScreenCoord	ScreenFocus
LLADegreesMeters	WorldFocus

◆ P3D::IVRTrackedControllerV450

class P3D::IVRTrackedControllerV450

Interface for accessing Virtual Reality Tracked Controller Information

Inherits IUnknown.

Private Member Functions

```
virtual ObjectLocalTransform & GetLocalTransform () override
virtual ObjectWorldTransform & GetWorldTransform () override
virtual VRControllerButton & GetButtonByType (VRControllerButtonType eType)
    override
virtual VRControllerAxis & GetAxisByType (VRControllerAxisType eType) override
```

Member Function Documentation

◆ GetAxisByType()

virtual VRControllerAxis&
GetAxisByType (VRControllerAxisType eType) private pure virtual

Returns the axis of the indicated axis type

Parameters

[in] **eType** Enum for the Axis Type

◆ GetButtonByType()

virtual VRControllerButton&
GetButtonByType (VRControllerButtonType eType) private pure virtual

Returns the Button of the indicated button type

Parameters

[in] **eType** Enum for the Button Type

◆ GetLocalTransform()

virtual ObjectLocalTransform& GetLocalTransform () private pure virtual

Get the Local Transform of the Tracked Controller

◆ **GetWorldTransform()**

virtual **ObjectWorldTransform&** GetWorldTransform() **private** **pure virtual**

Get the World Transform of the Tracked Controller

◆ **P3D::IVRInterfaceV451**

class P3D::IVRInterfaceV451

Interface for Virtual Reality

Inherits IVRInterfaceV450.

Private Member Functions

virtual bool	GetIsPortalEditMode () override
virtual IWindowV440 *	GetWindow (VRView eView) override
virtual LPCWSTR	GetNameOfHmd () override
virtual float	GetEyepointZOffset () override
virtual void	GetHMDOrigin (ObjectWorldTransform &HMDWorld) override
virtual ICameraSystemV450 *	GetCameraSystem (VRView eView) override
virtual void	GetWorldTransform (VRView eView, ObjectWorldTransform &worldTrans) override
virtual void	GetLocalTransform (VRView eView, ObjectLocalTransform &localTrans) override
virtual void	SetIsPortalEditMode (bool bEdit) override
virtual void	SetHMDOrigin (ObjectWorldTransform &HMDWorld) override
virtual IVRTrackedControllerV450 *	GetController (VRController controller) override
virtual P3DEyeData &	GetEyeData (VREye eye) override
virtual P3DMath::float4x4 &	GetProjectionMatrix (VRView eView) override
virtual P3DMath::float4x4 &	GetViewMatrix (VRView eView) override

Member Function Documentation

◆ **GetCameraSystem()**

virtual ICameraSystemV450* GetCameraSystem (VRView eView) **private** **pure virtual**

Returns the Camera System Interface for the indicated Virtual Reality View

Parameters

[in] **eView** Enum for the Virtual Reality View

◆ **GetController()**

virtual **IVRTrackedControllerV450***

GetController

(**VRController** controller) **private** **pure virtual**

Returns the Tracked Controller Interface for the indicated controller

Parameters

[in] **controller** Enum for the controller

◆ **GetEyeData()**

virtual **P3DEyeData&** GetEyeData (**VREye** eye) **private** **pure virtual**

Returns the Eye Data of the indicated Eye including focus, openness, pupil diameter and gaze information

Parameters

[in] **eye** Enum for which eye Left or Right

◆ **GetEyepointZOffset()**

virtual float GetEyepointZOffset () **private** **pure virtual**

Get the current Z offset of the Eyes

◆ **GetHMDOrigin()**

virtual void GetHMDOrigin (**ObjectWorldTransform** & HMDWorld) **private** **pure virtual**

Get the World Transform of the headset origin

Parameters

[in

◆ **GetIsPortalEditMode()**

virtual bool GetIsPortalEditMode () **private** **pure virtual**

Check if Portals are being edited

◆ GetLocalTransform()

```
virtual void GetLocalTransform( VRView eView,  
                               ObjectLocalTransform & localTrans  
                           )
```

private pure virtual

Get the Local Transform of the indicated Virtual Reality View

Parameters

[in] **eView** The desired view
[in]

◆ GetNameOfHmd()

```
virtual LPCWSTR GetNameOfHmd( ) private pure virtual
```

Returns the name of the Headset that is currently active

◆ GetProjectionMatrix()

```
virtual P3DMath::float4x4& GetProjectionMatrix( VRView eView ) private pure virtual
```

Returns the Projection Matrix for the indicated Virtual Reality View

Parameters

[in] **eView** Enum for the desired view

◆ GetViewMatrix()

```
virtual P3DMath::float4x4& GetViewMatrix( VRView eView ) private pure virtual
```

Returns the View Matrix for the indicated Virtual Reality View

Parameters

[in] **eView** Enum for the desired view

◆ GetWindow()

```
virtual IWindowV440* GetWindow( VRView eView ) private pure virtual
```

Returns the Window Interface of the indicated View

Parameters

[in] **eView** Enum for the Virtual Reality View

◆ GetWorldTransform()

```
virtual void GetWorldTransform( VRView          eView,  
                               ObjectWorldTransform & worldTrans  
                           )
```

private pure virtual

Get the World Transform of the indicated Virtual Reality View

Parameters

[in] **eView** The desired view
[in]

◆ SetHMDOrigin()

```
virtual void SetHMDOrigin( ObjectWorldTransform & HMDWorld )
```

private pure virtual

Set the World Transform of the headset origin

Parameters

[in] **HMDWorld** World Location to set the origin

◆ SetIsPortalEditMode()

```
virtual void SetIsPortalEditMode( bool bEdit )
```

private pure virtual

Sets the Portal editing mode

Parameters

[in] **bEdit** Boolean for the portal edit mode

◆ P3D::IVRSettingsV500

class P3D::IVRSettingsV500

Interface for accessing the Virtual Reality Settings

Inherits IVRSettingsV452.

Private Member Functions

```
virtual HMD_DISPLAY_MODE GetHMDDisplayMode () override
```

```
virtual HMD_REFERENCE_ORIGIN GetHMDReferenceOrigin () override
```

```
virtual float GetEyePointZOffset () override
```

```
virtual bool GetEnableZoom () override
```

```
virtual bool GetEnableEyeTracking () override
```

	virtual bool	GetEnableGazeSelection () override
	virtual bool	GetEnableToolTips () override
	virtual bool	GetEnableGazeDetection () override
	virtual bool	GetEnableEyeData () override
	virtual int	GetPassThroughCameraBrightness () override PassThroughCamera settings ///. More...
	virtual int	GetPassThroughCameraContrast () override
	virtual int	GetPassThroughCameraHue () override
	virtual int	GetPassThroughCameraSaturation () override
	virtual int	GetPassThroughCameraGain () override
	virtual int	GetPassThroughCameraExposure () override
	virtual int	GetPassThroughCameraWhiteBalance () override
	virtual int	GetPassThroughCameraResolution () override
	virtual int	GetPassThroughCameraMode () override
	virtual int	GetEnableGreenScreen () override
virtual P3DMath::float2 &	GetPassThroughCameraFOV () override	
	virtual int	GetPassThroughCameraSensorMode () override
	virtual bool	GetPassThroughCameraAutoExposure () override
	virtual bool	GetPassThroughCameraAutoWhiteBalance () override
	virtual float	GetPassThroughCameraExposureTime () override
	virtual float	GetPassThroughCameraGlobalGain () override
virtual P3DMath::float3 &	GetPassThroughCameraColorGain () override	
virtual PCWSTR	GetCameraEmulatorTexture () override	
	virtual int	GetVarjoViewMode () override
	virtual int	GetVarjoSyncMode () override

Member Function Documentation

◆ **GetCameraEmulatorTexture()**

virtual PCWSTR GetCameraEmulatorTexture () private pure virtual

Returns: A string representing the texture

◆ **GetEnableEyeData()**

virtual bool GetEnableEyeData () private pure virtual

Returns if Eye Data Panel is enabled

◆ GetEnableEyeTracking()

virtual bool GetEnableEyeTracking() **private** **pure virtual**

Returns whether eye tracking is enabled or not

◆ GetEnableGazeDetection()

virtual bool GetEnableGazeDetection() **private** **pure virtual**

Returns if Gaze Detection is enabled

◆ GetEnableGazeSelection()

virtual bool GetEnableGazeSelection() **private** **pure virtual**

Returns if Gaze Selection is enabled

◆ GetEnableGreenScreen()

virtual int GetEnableGreenScreen() **private** **pure virtual**

Returns: If green screen is enabled Default: 0

◆ GetEnableToolTips()

virtual bool GetEnableToolTips() **private** **pure virtual**

Returns if tooltips are enabled

◆ GetEnableZoom()

virtual bool GetEnableZoom() **private** **pure virtual**

Returns if zoom is enabled on the headset

◆ GetEyePointZOffset()

virtual float GetEyePointZOffset() **private** **pure virtual**

Returns the Button of the indicated button type

◆ GetHMDDisplayMode()

virtual **HMD_DISPLAY_MODE** GetHMDDisplayMode() **private** **pure virtual**

Get the current render mode setting

◆ GetHMDReferenceOrigin()

virtual **HMD_REFERENCE_ORIGIN** GetHMDReferenceOrigin() **private** **pure virtual**

Get the current reference origin setting

◆ GetPassThroughCameraAutoExposure()

virtual bool GetPassThroughCameraAutoExposure() **private** **pure virtual**

Returns: If the auto exposure is activated Default: 1

◆ GetPassThroughCameraAutoWhiteBalance()

virtual bool GetPassThroughCameraAutoWhiteBalance() **private** **pure virtual**

Returns: If the auto white balance is activated Default: 1

◆ GetPassThroughCameraBrightness()

virtual int GetPassThroughCameraBrightness() **private** **pure virtual**

PassThroughCamera settings ///.

Returns: The bightness setting Range: 0 - 8 Default: 4

◆ GetPassThroughCameraColorGain()

virtual **P3DMath::float3&** GetPassThroughCameraColorGain() **private** **pure virtual**

Returns: The color of the global gain Default: {1, 1, 1}

◆ GetPassThroughCameraContrast()

virtual int GetPassThroughCameraContrast() **private** **pure virtual**

Returns: The contrast setting Range: 0 - 8 Default: 4

◆ **GetPassThroughCameraExposure()**

virtual int GetPassThroughCameraExposure() **private** **pure virtual**

Returns: The exposure setting Range: 0 - 100 Default: 50

◆ **GetPassThroughCameraExposureTime()**

virtual float GetPassThroughCameraExposureTime() **private** **pure virtual**

Returns: The exposure time Default: 0

◆ **GetPassThroughCameraFOV()**

virtual **P3DMath::float2&** GetPassThroughCameraFOV() **private** **pure virtual**

Returns: The field of view Default: 85, 60 Default aspect ratio: 17:12

◆ **GetPassThroughCameraGain()**

virtual int GetPassThroughCameraGain() **private** **pure virtual**

Returns: The gain setting Range: 0 - 100 Default: 50

◆ **GetPassThroughCameraGlobalGain()**

virtual float GetPassThroughCameraGlobalGain() **private** **pure virtual**

Returns: The global gain Range: 0 - 1 Default: 1

◆ **GetPassThroughCameraHue()**

virtual int GetPassThroughCameraHue() **private** **pure virtual**

Returns: The hue setting Range: 0 - 11 Default: 0

◆ **GetPassThroughCameraMode()**

virtual int GetPassThroughCameraMode() **private** **pure virtual**

Returns: The camera mode Modes: Camera only, Portals, Inverted Portals, Green Screen,

Green Screen and Portals, Green Screen and Inverted Portals Default: 0 or Camera only

◆ **GetPassThroughCameraResolution()**

virtual int GetPassThroughCameraResolution () **private** **pure virtual**

Returns: The camera resolution setting Resolutions: 2K, 1080p, 720p, VGA Default: 720p

◆ **GetPassThroughCameraSaturation()**

virtual int GetPassThroughCameraSaturation () **private** **pure virtual**

Returns: The saturation setting Range: 0 - 8 Default: 4

◆ **GetPassThroughCameraSensorMode()**

virtual int GetPassThroughCameraSensorMode () **private** **pure virtual**

Returns: The sensor mode Modes: Left eye, Right eye, Both Default: 2 or Both

◆ **GetPassThroughCameraWhiteBalance()**

virtual int GetPassThroughCameraWhiteBalance () **private** **pure virtual**

Returns: The white balance setting Range: 2800 - 6500 or cool - warm Default: 4800

◆ **GetVarjoSyncMode()**

virtual int GetVarjoSyncMode () **private** **pure virtual**

◆ **GetVarjoViewMode()**

virtual int GetVarjoViewMode () **private** **pure virtual**

◆ **P3D::IVRServiceV452**

class P3D::IVRServiceV452

Inherits IVRServiceV450.

Private Member Functions

```
virtual void RegisterCallback (ICallbackV400 *pCallback) override  
virtual void UnregisterCallback (ICallbackV400 *pCallback) override  
virtual void UnregisterStereoCamera (IStereoCameraV500 *pCamera)  
override  
virtual void RegisterStereoCamera (IStereoCameraV500 *pCamera) override  
virtual IVRInterfaceV450 * GetVRInterface ()=0  
virtual IVRSettingsV450 * GetVRSettings ()=0  
virtual void EnableVR (HMD_INTERFACES eInterface)=0  
virtual void DisableVR ()=0
```

Member Function Documentation

◆ DisableVR()

```
virtual void DisableVR ( ) private pure virtual
```

Disables Virtual Reality

◆ EnableVR()

```
virtual void EnableVR ( HMD_INTERFACES eInterface ) private pure virtual
```

Enables Virtual Reality with the indicated HMD Device

Note

, If Virtual Reality is already enabled this function does nothing

Parameters

[in] **eInterface** Enum for the HMD Device

◆ GetVRInterface()

```
virtual IVRInterfaceV450* GetVRInterface ( ) private pure virtual
```

Returns the Virtual Reality Interface

◆ GetVRSettings()

```
virtual IVRSettingsV450* GetVRSettings ( ) private pure virtual
```

Returns the Virtual Reality Settings Interface

◆ RegisterCallback()

virtual void RegisterCallback (**ICallbackV400** * pCallback) **private** **pure virtual**

◆ RegisterStereoCamera()

virtual void RegisterStereoCamera (**IStereoCameraV500** * pCamera) **private** **pure virtual**

◆ UnregisterCallback()

virtual void UnregisterCallback (**ICallbackV400** * pCallback) **private** **pure virtual**

◆ UnregisterStereoCamera()

virtual void UnregisterStereoCamera (**IStereoCameraV500** * pCamera) **private** **pure virtual**

Typedefs

typedef **IVRInterfaceV451** **IVRInterface**

typedef **IVRSettingsV500** **IVRSettings**

typedef **IVRTrackedControllerV450** **IVRTrackedController**

typedef **IVRServiceV452** **IVRService**

Enumerations

enum **VRController** { **VR_CONTROLLER_LEFT**, **VR_CONTROLLER_RIGHT** }

enum **VRControllerButtonType** {
 BUTTON_TRIGGER, **BUTTON_DPAD_RIGHT**, **BUTTON_DPAD_LEFT**,
BUTTON_DPAD_DOWN,
 BUTTON_DPAD_UP, **BUTTON_MENU**, **BUTTON_GRIP**, **BUTTON_ANALOG**,
MAX_BUTTON_COUNT
}

enum **VRControllerAxisType** {
 AXIS_NONE, **AXIS_TOUCHPAD**, **AXIS_ANALOG**, **AXIS_TRIGGER**,
AXIS_DPAD, **MAX_AXIS_COUNT**
}

enum **HMD_DISPLAY_MODE** { **MONO** = 0, **STEREO** = 1, **SINGLEPASS** = 2 }

enum **HMD_REFERENCE_ORIGIN** { **REF_ORIGIN_HMD_ZERO** = 0,
REF_ORIGIN_P3D_CUSTOM, **REF_ORIGIN_P3D_TRACKED**, **REF_ORIGIN_MAX** }

enum **VREye** { **VR_EYE_LEFT**, **VR_EYE_RIGHT** }

enum **VRView** {
 VR_VIEW_LEFT, **VR_VIEW_RIGHT**, **VR_VIEW_LEFT_FOCUS**,
VR_VIEW_RIGHT_FOCUS,
 VR_VIEW_CENTER, **VR_VIEW_COUNT**
}

Variables

DEFIID IID_IVRInterfaceV451 = **IIDofIVRInterfaceV451**

```
REFIID IID_IVRInterfaceV451 = __uuidof(IVRInterfaceV451)
REFIID IID_IVRSettingsV500 = __uuidof(IVRSettingsV500)
REFIID IID_IVRTrackedControllerV450 = __uuidof(IVRTrackedControllerV450)
REFIID IID_IVRInterface = IID_IVRInterfaceV451
REFIID IID_IVRSettings = IID_IVRSettingsV500
REFIID IID_IVRTrackedController = IID_IVRTrackedControllerV450
REFIID IID_IVRServiceV452 = __uuidof(IVRServiceV452)
REFGUID SID_VRService = __uuidof(IVRServiceV450)
REFIID IID_IVRService = IID_IVRServiceV452
```

Typedef Documentation

◆ IVRInterface

```
typedef IVRInterfaceV451 IVRInterface
```

◆ IVRService

```
typedef IVRServiceV452 IVRService
```

◆ IVRSettings

```
typedef IVRSettingsV500 IVRSettings
```

◆ IVRTrackedController

```
typedef IVRTrackedControllerV450 IVRTrackedController
```

Enumeration Type Documentation

◆ HMD_DISPLAY_MODE

```
enum HMD_DISPLAY_MODE
```

Enumerator

MONO	
STEREO	
SINGLEPASS	

◆ HMD_REFERENCE_ORIGIN

enum HMD_REFERENCE_ORIGIN

Enumerator

REF_ORIGIN_HMD_ZERO
REF_ORIGIN_P3D_CUSTOM
REF_ORIGIN_P3D_TRACKED
REF_ORIGIN_MAX

◆ VRController

enum VRController

Enumerator

VR_CONTROLLER_LEFT
VR_CONTROLLER_RIGHT

◆ VRControllerAxisType

enum VRControllerAxisType

Enumerator

AXIS_NONE
AXIS_TOUCHPAD
AXIS_ANALOG
AXIS_TRIGGER
AXIS_DPAD
MAX_AXIS_COUNT

◆ VRControllerButtonType

enum VRControllerButtonType

Enumerator

BUTTON_TRIGGER
BUTTON_DPAD_RIGHT
BUTTON_DPAD_LEFT
BUTTON_DPAD_DOWN
BUTTON_DPAD_UP
BUTTON_MENU
BUTTON_GRIP
BUTTON_ANALOG
MAX_BUTTON_COUNT

◆ VREye

enum VREye

Enumerator

VR_EYE_LEFT

VR_EYE_RIGHT

◆ VRView

enum VRView

Enumerator

VR_VIEW_LEFT

VR_VIEW_RIGHT

VR_VIEW_LEFT_FOCUS

VR_VIEW_RIGHT_FOCUS

VR_VIEW_CENTER

VR_VIEW_COUNT

Variable Documentation

◆ IID_IVRInterface

REFIID IID_IVRInterface = **IID_IVRInterfaceV451**

◆ IID_IVRInterfaceV451

REFIID IID_IVRInterfaceV451 = __uuidof(**IVRInterfaceV451**)

◆ IID_IVRService

REFIID IID_IVRService = **IID_IVRServiceV452**

◆ IID_IVRServiceV452

REFIID IID_IVRServiceV452 = __uuidof(**IVRServiceV452**)

◆ IID_IVRSettings

REFIID IID_IVRSettings = **IID_IVRSettingsV500**

◆ IID_IVRSettingsV500

REFIID IID_IVRSettingsV500 = __uuidof(**IVRSettingsV500**)

◆ **IID_IVRTrackedController**

REFIID IID_IVRTrackedController = **IID_IVRTrackedControllerV450**

◆ **IID_IVRTrackedControllerV450**

REFIID IID_IVRTrackedControllerV450 = __uuidof(**IVRTrackedControllerV450**)

◆ **SID_VRService**

REFGUID SID_VRService = __uuidof(IVRServiceV450)

- top -

Related Links

- [Data Load Helper](#)
- [Visual Effects Service](#)
- [Event Service](#)
- [Global Data Service](#)
- [Rendering Services](#)
- [Window and Camera Services](#)
- [Weather System Service](#)
- [ISimObjects](#)
- [PDK Services](#)
- [PDK Types](#)
- [SDK Overview](#)
- [PDK Overview](#)

World Object Service

Overview

This PDK service allows callers to create, move, and delete objects in the world.

Classes

class **IWorldObjectServiceV450**

Class Documentation

◆ **P3D::IWorldObjectServiceV450**

class P3D::IWorldObjectServiceV450

Inherits IUnknown.

Private Member Functions

virtual HRESULT **CreateObject** (_in const GUID &guid, _in const **ObjectWorldTransform** &location, _in FLOAT scale, _in BOOL isOnGround, _out UINT &idObject) override

virtual HRESULT **MoveObject** (_in UINT idObject, _in const **ObjectWorldTransform** &location, _in BOOL isOnGround) override

virtual HRESULT **RemoveObject** (_in UINT idObject) override

Member Function Documentation

◆ **CreateObject()**

virtual HRESULT

```
CreateObject( __in const GUID & guid,  
           __in const ObjectWorldTransform & location,  
           __in FLOAT scale,  
           __in BOOL isOnGround,  
           __out UINT & idObject  
)
```

private **pure virtual**

Creates a world object by model GUID. Returns S_OK upon success, E_FAIL upon failure.

Parameters

- guid** The model GUID of the object to be created.
- location** The location that the model should be placed.
- scale** The amount that the model should be scaled. 1.0 is the default scale.
- isOnGround** Enable to ignore the specified altitude and get pulled to the ground.
- idObject** Object ID of the newly created object.

◆ MoveObject()

```
virtual HRESULT MoveObject( __in UINT idObject,  
                           __in const ObjectWorldTransform & location,  
                           __in BOOL isOnGround  
)
```

private **pure virtual**

Moves a world object by object ID. Returns S_OK upon success, E_FAIL upon failure.

Parameters

- idObject** The object ID of the object to be moved.
- location** The location that the model should be moved.
- isOnGround** Enable to ignore the specified altitude and get pulled to the ground.

◆ RemoveObject()

```
virtual HRESULT RemoveObject( __in UINT idObject )
```

private **pure virtual**

Removes a world object by object ID. Returns S_OK upon success, E_FAIL upon failure.

Parameters

- idObject** The object ID of the object to be removed.

Variables

GUID **IID_IWorldObjectServiceV450**
WorldObjectService interface ID. More...

GUID **SID_WorldObjectService**
WorldObjectService service ID. More...

Variable Documentation

◆ **IID_IWorldObjectServiceV450**

GUID IID_IWorldObjectServiceV450

WorldObjectService interface ID.

◆ **SID_WorldObjectService**

GUID SID_WorldObjectService

WorldObjectService service ID.

- top -

PREPAR3D

SimObject API

Overview

The SimObject API utilizes a service-based methodology for building simulation behaviors to be visualized in *Prepar3D*. The API enables a solution developer to create a simulation object (*SimObject*) complete with customized behaviors, input properties (also referred to as *events* or *triggers*), and state properties (also referred to as *simvars* or simply *properties*). These properties can be referenced in content such as SimObject gauges, animations, and scenario scripts which are discussed in more detail in other parts of the SDK. The properties are text-based, and can be referenced in the same way as the stock *simvars* and *events* are referred to in other parts of the Prepar3D SDK.

GettingStarted

A solution developer using this SDK should be familiar with the following:

- **IUnknown** and **IServiceProvider**. These standard COM interfaces are utilized throughout the SDK. These are available in a typical *Microsoft Visual Studio* installation using a header file such as *atlcomcli.h*.
- All interfaces obtained through these interfaces are reference-counted. It is important to understand the importance of releasing references to these objects to avoid memory leaking.
- All the sample solutions in this SDK were created with *Microsoft Visual Studio 2019*.
- SimObject API implementations are loaded into Prepar3D by loading a custom DLL at application load time. Instructions on loading a custom DLL can be found in the [SimConnect Overview](#).
- The contents of the relevant header files for this SDK are defined within the *P3D* namespace. For convenience, the samples all use the declaration "using namespace *P3D*".

SimObject Conventions and Standards

Prepar3D SimObjects must obey the following conventions to function correctly:

- Left-Handed axes:
 - X axis: Lateral, positive right
 - Y axis: Vertical, positive up
 - Z axis: Longitudinal, positive forward
 - Pitch rotation, positive downward about X axis
 - Yaw/Heading rotation, positive right about Y axis
 - Roll/Bank rotation, positive left about Z axis
- Units. Unless stated otherwise in the variable or interface name, standard English units should be assumed:
 - Length: feet
 - Weight: pounds-force
 - Mass: slugs
 - Time: seconds
 - Speed: feet / second
 - Acceleration: feet / second²
 - Angle: radians
 - Temperature: degrees Rankine
- Earth Coordinates
 - Ellipsoid model
 - Constant polar and equatorial radii
 - Altitude referenced from Mean Sea Level in feet, positive up
 - Latitude referenced from the equator in radians, positive North
 - Longitude referenced from the prime meridian, positive East

SimObject API Samples

The Prepar3D SDK comes with several samples that show the functionality of the ISimObject API. More information is available on the [PDK Samples Overview](#) page.

CONTENTS

- [Getting Started](#)
- [SimObject Conventions and Standards](#)
- [SimObject API Samples](#)
- [Creating a new solution and DLL using the SimObject API](#)
 - [Create new VC++ project](#)
 - [Configuring your DLL](#)
 - [Creating a New SimObject Class](#)
 - [Registering Your SimObject Implementation](#)
- [SimObject Instance Lifecycle](#)
 - [Creation](#)
 - [Loading Constant Data](#)
 - [Loading Dynamic Data](#)
 - [Initialization](#)
 - [De-Initialization](#)
 - [Unloading Constant Data](#)
 - [Destruction](#)
- [Creating Behaviors](#)
 - [Registering Simulation Callbacks](#)
 - [Saving Simulation States](#)
- [Creating Properties and Events](#)
 - [Properties](#)
 - [Events](#)

RELATED LINKS

- [SDK Overview](#)
- [SimObject API References](#)
- [SimObject API Content Creation](#)
- [PDK Samples Overview](#)
- [Aircraft Configuration Files](#)
- [Add-ons Overview](#)
- [SimDirector](#)
- [Programmable Gauges](#)
- [New Aircraft Procedures](#)

Creating a new solution and DLL using the SimObject API

Create new VC++ Project

- New Project - Visual C++ - Empty Project
- Add a .cpp source file
- Add a .Def Module Definition File
- Set Configuration Properties
 - General
 - Type: Dynamic Library (.dll)
 - Target Extension: .dll
 - Character Set: Use Multi-Byte Character Set
 - C/C++ / General
 - Additional Include Directories: Add path to:
 - <Prepar3D SDK Path>\inc\PDK\
 - C/C++ / Advanced
 - Calling Convention: __stdcall (/Gz)
 - Linker, Input
 - Module Definition File: Set your .def file.
 - AfterBuild Target
 - Create an AfterBuild target to copy all content to the output directory.
Each sample has an example of this command in each .vcxproj file.
- Add an add-on.xml file pointing to the built .dll and any necessary content. Each sample has an example. See [Add-on Packages](#) for details on this process.

Configuring your DLL

- Implement the DLL start/stop functions that Prepar3D will call
 - void __stdcall DLLStart(PDK::IPdk* pPdk) {}
 - void __stdcall DLLStop() {}
- Note that DLLStart() and DLLStop() are necessary for Prepar3D to validate your DLL even if they have no functionality in the body.
- Export these in your .DEF file (See sample .def)
- At this point, Prepar3D should be able to load your successfully built .dll.

Creating a New SimObject Class

- To register a new SimObject class, you must implement three components:
- The implementation class derived from ISimObject.
- A static "factory" function used to create instances of your SimObject implementation. The address of this function will be passed as a function pointer using the function pointer prototype PSimCreateFunc found in ISimObject.h.
- A unique GUID id for your implementation class.

For starters, this can be as simple as the following example:

```
#include <atlcomcli.h>
#include <ISimObject.h>
#include <InitGuid.h>

class MySimObject : public P3D::ISimObject
{
    STDMETHOD (LoadConstantData) (_out void** ppConstantData)
    STDMETHOD (UnloadConstantData) (_inout void** ppConstantData)
    STDMETHOD (LoadDynamicData) ()
    STDMETHOD (Init) ()
    STDMETHOD (DeInit) ()

    // IUnknown functions will be required here
};

static HRESULT New(_in _notnull P3D::IBaseObject* pBaseObject,
    _out _notnull P3D::ISimObject** ppThisSim)
{
    return E_FAIL;
}

DEFINE_GUID(CLSID_MySimObject, 0x2afb2ae8, 0xe74, 0x4d76, 0x8c,
0x6e, 0x7b, 0x5a, 0x42, 0xef, 0x19, 0x6e);
```

Registering Your SimObject Implementation

To register your SimObject class, you must first obtain a reference to the *Prepar3D Developer Kit* interface. The PDK API is a service provider for obtaining services to the Prepar3D platform. An interface to the PDK (*IPdk*) will be necessary to obtain the SimObject Manager in order to register a new SimObject implementation. Like the other interfaces in this SDK, the PDK is a reference-counted object so the reference must be released to avoid memory leaks. Smart pointers are also convenient for this purpose and are used extensively in the samples. Here is an example of how to register the above SimObject implementation:

Registration with the SimObject Manager object requires 3 elements. The unique GUID and static "factory" function were discussed in the previous section. In addition, a friendly category name must be specified. This is used for filtering objects of similar classes, such as "Airplane", "Helicopter", or "GroundVehicle". This name is also referenced in the Prepar3D.cfg [Main] User Objects setting to determine which object categories are selectable through the UI.

```
#include <atlcomcli.h>
#include <Pdk.h>
#include <ISimObject.h>
```

```

void __stdcall DLLStart(P3D::IPdk* pPdk)
{
    P3D::ISimObjectManager* pManager = NULL;

    HRESULT hr = E_FAIL;

    if (pPdk)
    {
        //Get the SimObject Manager from the PDK Service
        hr = pPdk->
>QueryService(P3D::SID_SimObjectManager, P3D::IID_ISimObjectManager, (void**)&pManager);

        if (SUCCEEDED(hr))
        {
            //Register your SimObject class with the SimObject Manager with its unique guid ID, friendly
            //category name, and address for its "create" function.
            hr = pManager->
>RegisterSimulationCategory(CLSID_MySimObject,           //Unique GUID
                                "Airplane",                //Friendly category
                                New);                      //Static "factory" function

        }
    }

    pManager->Release(); //Don't forget to release reference
    pPdk->Release();    //Don't forget to release reference
}

```

At this point, Prepar3D will load your DLL and register your SimObject implementation. It is important to note that this registration should occur in the call to your `DLLStart()`, which occurs early in the Prepar3D loading process. Shortly after that, validation occurs on all SimObjects against their specified simulation implementation.

SimObject Instance Lifecycle

The following sections will illustrate the lifecycle of a SimObject instance: creation, initialization, and destruction.

Note: The following code samples are meant to be general examples, and will not necessarily compile with the previous code without additional implementations, such as the `IUnknown` methods. The samples provided in this SDK will provide more thorough functioning code.

Creation

A SimObject instance is created when Prepar3D calls your factory function that was registered in the preceding steps. Prepar3D will call your function with an `IBaseObject` interface pointer. The base object is basically a proxy object that manages data between your SimObject implementation and other systems throughout Prepar3D. Your function must return a new instance of your `ISimObject` in the `out`-parameter.

Note:

- Both the base object must abide by the rules of `IUnknown`, including reference-counting.
- This `IBaseObject` reference will be used to register behaviors and properties, as well as real-time data transfer with the base object.
- The three `IUnknown` functions must be added to your SimObject class in order to instantiate it. See the sample for examples how to do this.
 - `AddRef()`
 - `Release()`
 - `QueryInterface()`

Following is an example of a factory function. This would require a constructor to be added to your object class.

```

static HRESULT New(__in __notnull P3D::IBaseObject* pBaseObject,
__out __notnull P3D::ISimObject** ppThisSim)
{
    *ppThisSim = new MySimObject(pBaseObject);

    return S_OK;
}

```

Loading Constant Data

Once your basic object is instantiated, Prepar3D will call your `ISimObject` implementation of `LoadConstantData()`. For performance reasons, Prepar3D will cache the data returned in the `out`-parameter and used for subsequent instances of the same SimObject. If the data exists for the new SimObject, Prepar3D will pass it in so that the same data can be used without having to reload it. It is important to note that this is only for data that can be shared across all instances of the same SimObject.

```

STDMETHODIMP MySimObject::LoadConstantData(__out void** ppConstantData) //on-disk data
{
    HRESULT hr = E_FAIL;

    //Do we already have a ref to loaded data due to a previous
    instance?
}

```

```

    if (*ppConstantData)
    {
        //Use this data
        m_pConstantData =
reinterpret_cast<MySimObjectConstantData*>(*ppConstantData);
    }
    else //Else, Load the data, return a ref to it to be cached
for subsequent instances
    {
        //Load data and return is
        m_pFixedData = new MySimObjectConstantData();

        hr = m_pConstantData->Load();

        *ppConstantData =
const_cast<void*>(reinterpret_cast<const
void*>(m_pConstantData));
    }

    return hr;
}

```

Loading Dynamic Data

In this loading phase, your SimObject may create subsystems that may be dependent on the constant data loaded in the previous phase. An example might be instantiating the number of engines on an airplane based on the data loaded in the previous phase.

```

STDMETHODIMP MySimObject::LoadDynamicData()
{
    HRESULT hr = E_FAIL;

    //Create your SimObject subsystems
    hr = S_OK;

    return hr;
}

```

Initialization

In this initialization phase, your SimObject should have all of its subsystems created. It might be used to initialize the state of a system or establish reference to other systems. An example might be references between an engine and a fuel system.

```

STDMETHODIMP MySimObject::Init()
{
    HRESULT hr = E_FAIL;

    //Initialize your subsystems
    hr = S_OK;

    return hr;
}

```

De-initialization

In this phase, your SimObject might need to release references between dependent subsystems.

```

STDMETHODIMP MySimObject::DeInit()
{
    HRESULT hr = E_FAIL;

    //De-initialize your subsystems
    hr = S_OK;

    return hr;
}

```

Unloading Constant Data

When the last instance of a SimObject is destroyed, Prepar3D will call UnLoadConstantData() to destroy it:

```

STDMETHODIMP MySimObject::UnLoadConstantData(__in void**
ppConstantData) //on-disk data
{
    HRESULT hr = E_FAIL;

    if (*ppConstantData == m_pConstantData)
    {
        hr = m_pConstantData->UnLoad();

        delete m_pConstantData;

        m_pConstantData = NULL;
        *ppConstantData = NULL;

        hr = S_OK;
    }

    return hr;
}

```

Destruction

Destruction will occur when the reference count on your SimObject reaches zero.

Creating Behaviors

Registering Simulation Callbacks

Real-time simulation callbacks are registered through the [IBaseObject](#) interface using the [ISimulation](#) interface. For example:

```
class MySimulation : public P3D::ISimulation
{
    STDMETHOD (Update)(double dDeltaT) override { /*Do simulation
stuff*/ return S_OK; }
    STDMETHOD (SaveLoadState)(__in __notnull
P3D::PSaveLoadCallback pfnCallback, __in BOOL bSave) override
    {
        if (bSave)
        {
            //Save states
        }
        else
        {
            //Load states
        }
    }
}

//SimObject definition
class MySimObject : public P3D::ISimObject
{
    ...
    MySimulation m_MySimulation;//Instance of MySimulation
}

//Function called to register simulation callbacks
void MySimObject::RegisterMySimulations()
{
    HRESULT hr = pBaseObject->RegisterSimulation(&m_MySimulation,
60.0f /*Hz*/);
}
```

Useful notes on implementing [ISimulation](#) classes:

- *ISimulation* is derived from *IUnknown*, which will require implementation of the *AddRef()*, *Release()*, and *QueryInterface()*. This will allow clean interfacing between subsystems.
- Registration requires a simulation rate in *hertz*.
- Prepar3D does not limit the number of *ISimulation* callbacks that are registered.

Saving Simulation States

SaveLoadState() is called on all *ISimulation* implementations when Prepar3D executes both a Scenario Save and Scenario Load.

- The *BOOL* argument corresponds to whether the state is being saved (TRUE) or loaded (FALSE).
- In either case, the function pointer *PSaveLoadCallback* allows your code to pass in parameters corresponding to the .FXML data formats. For example:

```
STDMETHODIMP MySimObject::SaveLoadState(__in __notnull
P3D::PSaveLoadCallback pfnSaveLoadCallback, __in const BOOL
bSave)
{
    if (pfnSaveLoadCallback)
    {
        /*Section
Name*/ /*Instance*/ /*Keyword*/ /*Value*/
        pfnSaveLoadCallback("MySimObjectData",      0      ,
"MyDefaultValue_0", m_dVal0, bSave);
        pfnSaveLoadCallback("MySimObjectData",      0      ,
"MyDefaultValue_1", m_dVal1, bSave);
    }

    return S_OK;
}
```

Creating Properties and Events

Properties

This SDK enables a SimObject solution developer to create custom "properties" for their SimObject implementations. These are similar in nature to legacy [Simulation Variables](#).

- Properties must be registered when your DLL loads.
- A property registration consists of an association of the following:
 - The unique GUID id used to register your simulation implementation
 - String name for the property
 - Units for the property. This should be whatever your implementation will return
 - A Static function address to process the property query using the function pointer prototypes: *PPropertyCallback* for doubles or

PPropertyVectorCallback for vectors using the DXYZ structure found in ISimObject.h

- Registration must occur after the SimObject class is registered with the SimObject Manager. See the example below.
- An index can be used in the query for multiple instances of a common property. For example, a single property registration could be used for engine 0 rpm, engine 1 rpm, etc...
- Properties can be queried for animations, gauges, and scripts in the same manner as the legacy Simulation Variables.
- Property names are limited to 64 characters

Example of static callbacks registered with RegisterProperty():

```
#include <atlcomcli.h>
#include <Pdk.h>
#include <ISimObject.h >

//The following 2 samples functions must be static

//Sample PPropertyCallback function
/*static*/ STDMETHODIMP MySimObject::GetMyDoubleProperty(_in
const ISimObject& Sim, __out double& dProperty, __in int iIndex)
{
    dProperty = static_cast<const
MySimObject&>(Sim).m_dMyDoublePropertyValue;

    return S_OK;
}

//Sample PPropertyVectorCallback function
/*static*/ STDMETHODIMP MySimObject::GetMyVectorProperty(_in
const ISimObject& Sim, __out DXYZ& vProperty, __in int iIndex)
{
    vProperty = static_cast<const
MySimObject&>(Sim).m_vMyVectorPropertyValue;

    return S_OK;
}

void __stdcall DLLStart(P3D::IPdk* pPdk)
{
    P3D::ISimObjectManager* pManager = NULL;

    HRESULT hr = E_FAIL;

    if (pPdk)
    {
        //Get the SimObject Manager from the PDK Service
        hr = pPdk->QueryService(P3D::SID_SimObjectManager,
P3D::IID_ISimObjectManager, (void**)&pManager);
        if (SUCCEEDED(hr))
        {
            //Register your SimObject class with the SimObject
Manager with its unique guid ID, friendly
            //category name, and address for its "create"
function.
            hr = pManager-
>RegisterSimulationCategory(CLSID_MySimObject, //Unique GID
                                         "Airplane",           //Friendly
category                               New);           //Static
            "factory" function

            //Sample registration of properties
            pSimObjectMgr->RegisterProperty(CLSID_MySimObject,
"MyDoubleProperty", "percent over 100",
MySimObject::GetMyDoubleProperty);
            pSimObjectMgr->RegisterProperty(CLSID_MySimObject,
"MyVectorProperty", "percent over 100",
MySimObject::GetMyVectorProperty);

        }
    }

    pManager->Release(); //Don't forget to release reference
    pPdk->Release();     //Don't forget to release reference
}
```

Events

Similar to property registration, events are simply another type of property. They are analogous to Prepar3D's legacy [Event IDs](#). The same guidelines listed above for properties also apply to events, with the following exceptions:

- The function prototype used is *PEventCallback*.
- Registration requires an addition enum to distinguish if the event will be associated with an axis control such as a joystick, a POV (also known as a "hat switch"), or normal button press in the Controls Assignment UI:
 - EVENTTYPE_NORMAL, EVENTTYPE_AXIS, EVENTTYPE_POV, or EVENTTYPE_NOT_MAPPABLE
 - Axis event data is in the range of +/-1.0.
 - POV event data is provided as direction of the switch (commonly known as a joystick hat switch.) One of eight discrete switch direction data is given in degrees: 0, 45, 90,... 315. Note: upon release of the POV switch, the event data will be "-1.0"

-
- An event that is not mappable simply will not show up in the Controls Assignment UI. These events are typically triggered through animation or gauge scripts.
 - The registered event callback utilizes an input argument of type double, although Prepar3D's existing controls system limits data to 16-bit integer values if being called through this system. This includes keyboard, joystick, and gauge systems.
 - The registered event callback utilizes an input argument for units, similar to properties above, although Prepar3D's existing controls system does not support this capability yet.
 - Event property names must be limited to 36 characters if they are intended to be used in a mouse rectangle trigger embedded in a visual model, such as a button in the Virtual Cockpit. This is a limitation of the .MDL format.

An example of event property registration looks similar to the properties registered above:

```
//The following 2 samples functions must be static

//Sample PEventCallback function
/*static*/ STDMETHODIMP MySimObject:: MyKeyEventTrigger(_in
ISimObject& Sim, __in double dProperty, __in int iIndex)
{
    static_cast<MySimObject>(Sim).m_bMyKeyState =
!static_cast<MySimObject>(Sim).m_bMyKeyState; //Toggle

    return S_OK;
}

//Sample PEventCallback function
/*static*/ STDMETHODIMP MySimObject::MyJoyStickEvent(_in
ISimObject& Sim, __in double dProperty, __in int iIndex)
{
    static_cast<const MySimObject>(Sim).m_dMyJoyStickEvent =
dProperty;
    return S_OK;
}

void __stdcall DLLStart(P3D::IPdk* pPdk)
{
    ...
    //Sample registration of event property

    pSimObjectMgr->RegisterProperty(CLSID_MySimObject,
"MyKeyEventProperty", "bool", MySimObject::MyKeyEventTrigger,
EVENTTYPE_NORMAL);
    pSimObjectMgr->RegisterProperty(CLSID_MySimObject,
"MyJoystickProperty", "percent over 100",
MySimObject::MyJoyStickEvent) , EVENTTYPE_AXIS;
}
```

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

Creating Content

SimObject API

Overview

Content generally refers to the files and data associated for a particular instance of a SimObject implementation. The following elements apply to your custom SimObject implementation consistent with Prepar3D's native implementations. Relevant information can be found in [Simulation Object Configuration Files](#), [Sound Configuration Files](#), [C++ Gauges Overview](#), [Camera Configuration](#), [Modeling](#), and [Special Effects](#).

- Visual Model
- Visual Effects
- Sound
- Panels and Gauges
- Cameras

Simulation Data

In the *[General]* section each of your SimObject's sim.cfg files, the *CategoryId* must be set, which is the GUID id used to register your implementation. See [Registering Your SimObject Implementation](#). From the SimpleCar sample sim.cfg:

```
[General]
CategoryId={3F3B2498-7E88-420B-A46D-C9EEF6BB59D0} ;Id for
SimpleCar implementation
```

Physical simulation data is associated with your implementation, thus is not relevant to this SDK documentation. References to physical simulation data in [Simulation Object Configuration Files](#), should be disregarded for your SimObject. Prepar3D will however use the *static_cg_height* and *static_pitch* values in the *[contact_points]* section. These values are used when initializing a SimObject on the ground. For example:

```
[contact_points]
static_cg_height = 2.6    //feet, altitude of CG when at rest on
the ground
static_pitch = 1.0        //degrees, pitch when at rest on the
ground_ (+=Up, -=Dn)
```

Configuring Controls

Custom events created for your SimObject implementation can be configured for joystick, keyboard, and mouse use. Each SimObject sim.cfg can specify its own default mappings. *ControlMapFile* references the file path containing the control maps. Although the samples include this file with the SimObject's container for convenience, it is not necessary and can exist anywhere and be shared across multiple SimObjects.

```
[General]
ControlMapName=SimpleCar
ControlMapFile=Controls\DefaultControlMap.xml
```

ControlMapName is the specific name Prepar3D associates with the mapping. Prepar3D ultimately saves its complete mapping to %APPDATA%\Lockheed Martin\Prepar3D v5\Controls\Standard.xml, and *ControlMapName* is appended to the standard mapping names for distinction. For example:

```
<SimControls.Map>
  <Name>KEYBOARD_MAIN.SimpleCar</Name>
  <Entry>
    <Key>W</Key>
    <Down>IncThrottle</Down>
  </Entry>
  <Entry>
    <Key>S</Key>
    <Down>DecThrottle</Down>
  </Entry>
</SimControls.Map>
<SimControls.Map>
  <Name>SideWinder Precision 2 Joystick.SimpleAirplane</Name>
```

RELATED LINKS

- [SDK Overview](#)
- [SimObject API Getting Started](#)
- [SimObject API Debugging](#)
- [SimObject API References](#)
- [Aircraft Configuration Files](#)
- [SimDirector](#)
- [Programmable Gauges](#)
- [New Aircraft Procedures](#)

```

<Axis>
  <AxName>Slider</AxName>
  <Index>6</Index>
  <AxEvent>Throttle</AxEvent>
  <AxScale>127</AxScale>
  <AxNull>1</AxNull>
</Axis>
</SimControls.Map>

```

These mappings have priority over Prepar3D's own key mappings when there are multiple events mapped to a common key or control. There are also two other things to take note of:

- Keyboard commands registered through the SimObject API will be reflected in the UI Control Settings (Options - Settings - Controls - Buttons and Keys) under the Event Category filter for Simulation Object Events. This list will update depending on the simulation object currently selected by the user.
- There is a subtle difference in how the mapping load between keyboard and axes commands. The custom simulation keyboard map is layered in with the Prepar3D defaults, while the joystick and mouse mappings for custom simulation objects are defined entirely by the custom mapping.
 - For keyboard commands in KEYBOARD_MAIN the Standard.xml loads in addition to the corresponding KEYBOARD_MAIN. CurrentSimObject defined by the current user simulation object.
 - The joystick and mouse mappings are completely customized for each simulation object implementation. This is because the joystick and mouse are completely reinitialized whenever the user's object is loaded, but the keyboard is not.

Configuring Sound

The Sound.cfg is used to configure the sounds produced by your simobject implementation. There are three basic categories of sound that are supported: Triggers, ground roll, and engine sounds.

See also: [Sound Configuration Files](#)

Trigger

Trigger sounds are new to the [ISimObject](#) interface. They work in conjunction with the TriggerSound() method in the [IBaseObject](#) interface.

```

[Trigger.0]           ;Triggers should be sequentially numbered
Name=EngineStart      ;The name referenced in TriggerSound(). See IBaseObject
Loop=0                ;0 = Single shot, 1 = Looping
InternalOnly=0         ;Is this sound only played when the user view is inside the object.
UiGroup=Engine         ;Which sound group?
(Engine, Cockpit, Internal, Environment, or Voice)
filename=StartPis     ;.wav or .ogg file name
minimum_volume=10000   ;Lowest sound level to be heard (decibels)
full_scale_distance=200 ;Distance (meters) in which sound begins to attenuate

```

Ground Roll

Ground interaction sounds work similar to the Prepar3D native aircraft (see [Sound Configuration Files](#)). Whereas native aircraft use a fixed set of contact point names for section names, for ISimObjects use CONTACT.0, CONTACT.1, ... CONTACT.n. GROUND_ROLL is also a valid section name. The keywords "Name" and "Type" are unique for ISimObjects:

Properties for ground roll:

Name - The name should be unique. This name will be referenced by [TriggerContactSound\(\)](#) in the [IBaseObject](#) interface. If no name is specified, the section name will be used by default.

Type - The type must be either "Impact" or "Scrape", depending on whether the sound is intended to play just on initial contact with the ground, or while it is moving on the ground, respectively.

Example:

```

[CONTACT.0]
Name=MrapWheel.0          ;Unique name
Type=Impact                ;Type of sound. Impact or Scrape.
filename=cmtouch1, cmtouch2, cmtouch3 ;sound files used

```

GROUND_ROLL is unique in that no Name should be specified. "GROUND_ROLL" will be the name specified in the [TriggerContactSound\(\)](#) call.

```

[GROUND_ROLL]
filename=cnroll2
flags=125218
minimum_volume=6000
maximum_volume=9200
minimum_speed=3
maximum_speed=55
minimum_rate=0.80
maximum_rate=1.60
link=GROUND_ROLL2

```

Engine Sounds

Engine sounds for [ISimObjects](#) work similar to the native aircraft implementations. See [Sound Configuration Files](#) for specifics on the first segment of [SOUND_ENGINE]

below. [ISimObjects](#) can configure engine sound, propeller sound, and rotor sound by specifying the property for the sound system to query at runtime:

Properties for basic engine sounds:

CombustionProperty - Should return a percentage value from 0 (no combustion) to 100% (max)

CombustionPowerProperty - Should return a percentage value between reflecting power output.

NonCombustionPowerProperty - Should return a percentage value between reflecting engine sound when not combusting. For example, engine off but spooling down.

StarterProperty - Should return a Boolean value reflecting the starter on / off

EngineOffsetProperty - Should return an XYZ value for the engine offset from the model center.

Properties for propeller sounds:

PropellerBetaAngleProperty - Should return the pitch angle of the propeller.

PropellerPercentRpmProperty - Should return the percentage of max RPM of the propeller.

PropellerMaxBetaProperty - Should return the maximum pitch angle of the propeller.

PropellerMinBetaProperty - Should return the minimum pitch angle of the propeller in non-reverse mode.

PropellerMinReverseBetaProperty - Should return the minimum pitch angle of the propeller in reverse mode.

Properties for helicopter rotor:

RotorPercentRpmProperty - Should return the percentage of maximum rated rotor RPM

Example:

```
[SOUND_ENGINE]
number_of_engines=1
engine_type=custom
eng1_combustion=COMBUSTION.1.00
eng1_starter=starter
eng1_combustion_start=combstart
eng1_shutdown.shutdown
xeng1_prop=PROP.1.00
eng1_non_combustion=NON_COMBUSTION.1.00
```

ISimObject Properties

The following specify the property name to query the respective variables:

```
CombustionProperty=Vehicle.Combustion
CombustionPowerProperty=Vehicle.PercentRpm
NonCombustionPowerProperty=Vehicle.PercentRpm
StarterProperty=Vehicle.StarterOn
EngineOffsetProperty=Vehicle.EngineOffset
```

Use the following when modeling a propeller:

```
PropellerBetaAngleProperty=
PropellerPercentRpmProperty=
PropellerMaxBetaProperty=
PropellerMinBetaProperty=
PropellerMinReverseBetaProperty=
```

Use the following when modeling a rotor:

```
RotorPercentRpmProperty=
```

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

Debugging

SimObject API

Overview

Logging

Basic logging for custom implementation can be turned on in your Prepar3D.cfg:

```
[Main]
SimObjectApiLogging=1
```

The log will be printed to %USERPROFILE%\Documents\Prepar3D v5 Files\SimObjectApi.log. Upon successful registration of your custom SimObject implementation, a section will be created with your implementation's unique guid. In addition, each successfully created property will be listed within that section.

RELATED LINKS

- [SDK Overview](#)
- [SimObject API Getting Started](#)
- [SimObject API Content Creation](#)
- [SimObject API References](#)
- [Aircraft Configuration Files](#)
- [SimDirector](#)
- [Programmable Gauges](#)
- [New Aircraft Procedures](#)

Property Display

Debug windows can be created to display real-time values. An XML file defining the window and properties can be pointed at in your Prepar3D.cfg. For example:

```
[Main]
SimObjectApiPropertyDebugger=SDK\Simulation
Objects\ISimObject\Samples\SimpleAirplane\PropertyDebugging.xml
```

Below is an example of a configured window.

- The CategoryId must match that of your SimObject implementation.
- The Windowtitle is optional
- Property names must match those registered.
- Units can be specified as desired.
- Type should be Double or Vector to match the registered property. Double is used by default.
- Multiple windows can be defined as desired.
- WindowWidth (pixels) is optional and used to override the default.
- Priority is an option to control window order when working with multiple debug windows. The higher the priority value, the higher the window layer will be. The highest priority window will be the topmost window. The default value is zero.
- See the sample for examples.

```
<Simvar.PropertyOutputWindow CategoryId="{85B2CD9D-8156-4C93-B069-61B2D0A96C12}" Windowtitle="Simple Airplane Misc"
WindowWidth="4500" Priority="2">
  <Property>
    <Name>Throttle</Name>
    <Type>Double</Type>
    <Units>Percent</Units>
  </Property>
  <Property>
    <Name>PctGearExtended</Name>
    <Type>Double</Type>
    <Units>Percent</Units>
  </Property>
</Simvar.PropertyOutputWindow>
```

- top -

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

PDK Samples Overview

The Prepar3D SDK comes with several samples that show the functionality of the PDK API. The samples provided are grouped by high level capabilities or use cases. SimObjects developed using the SimObject API portion of the PDK are referred to as ISimObjects.

For more information on developing using the PDK, please refer to the [PDK API Overview](#). For additional information ISimObject development, please refer to the [SimObject API Overview](#).

General PDK Samples

Each of the following samples are located in the (*PDK General Samples*) directory in the Prepar3D SDK:

CONTENTS

- [General PDK Samples](#)
- [ISimObject PDK Samples](#)
- [Running the PDK API Samples](#)
- [Understanding the PDK API Samples](#)
- [Using a SimObject API Sample as a Template](#)

RELATED LINKS

- [SDK Overview](#)
- [SimObject API References](#)
- [SimObject API Content Creation](#)
- [Aircraft Configuration Files](#)
- [Add-ons Overview](#)
- [SimDirector](#)
- [Programmable Gauges](#)
- [New Aircraft Procedures](#)

Sample	Description
CameraSystemPDK	Demonstrates how to access and control cameras using the PDK. Refer to the Camera Sample Overview for more information on this sample.
CigiComponentControl	Demonstrates how to control CIGI components using the PDK when Prepar3D is an active CIGI host.
CigiCustomPacket	Demonstrates how to configure custom CIGI packets.
CustomIcons	Demonstrates how the PDK can be used to create and render custom icons sets.
CustomLights	Demonstrates how to create and draw lights using the PDK. Refer to the Custom Lights Sample documentation for more details on this sample.
CustomPDKObjects	Demonstrates how to use the PDK to draw various shapes at specific locations.
DataHarvester	Demonstrates how the PDK can be used to request data on an object and output this to a file. In this case data for the user object is outputted to a CSV file.
EyeTracking	Example API to interface with eye tracking devices, and interacting with VR using eye tracking data. Can be used in conjunction with the eye tracking panel and HMD emulator to simulate eye tracking with no hardware.
GazeDataCapture	Demonstrates how the PDK can be used to listen for Virtual Reality Service Messages and render a sphere at the Eye Location.

HMDSample	Demonstrates how the PDK can be used to create a Helmet Mounted Display (HMD) overlay. Scaleform is used in this sample for drawing the overlay.
LinuxSimulationIntegration	Demonstrates how the PDK can be used to interface between a vehicle hosted on an Linux simulation system. Refer to the Linux Simulation Integration Sample documentation for more information.
MousePicking	Demonstrates how to use the PDK to hook up custom 3D picking within a window. Refer to the Picking Sample documentation for more information on this sample.
MyEngine	Demonstrates how the PDK can be used to create and override specific vehicle components. In this case the throttle control for an aircraft engine.
MySubsystem	Demonstrates how the PDK can be used to introduce a real-time subsystem into the simulation.
OpenGLTexture	Demonstrates how the PDK can be used to render an OpenGL texture to a panel.
RadarAltimeter	Demonstrates how the PDK can be used to create a Radar Altimeter.
RadarPanelCallbackSample	Demonstrates how the PDK can be used to create a radar panel. The gauge is written in XML. Refer to the Radar Panel Callback documentation for more information.
SetSimRates	Demonstrates how to update the sim rate of simulation objects. This can be used on both native SimObjects and ISimObjects created using the SimObject API.
TargetingPodSample	Demonstrates how the PDK can be used to create an embedded targeting pod in an aircraft. The gauge overlay is written in Scaleform. Refer to the Targeting Pod Overview documentation for more details on this sample.
TextureandEffectRender	Demonstrates how the PDK can be used to render a cursor into a gauge using a texture or effect.
Tracker	Demonstrates how to drive an ISimObject's position using an external real-time data source from the internet.
	Demonstrates how to

WorldObject	create, place, and move an object in the world.
--------------------	---

ISimObject PDK Samples

Each of the following samples are located in the (*PDK ISimObject Samples*) directory in the Prepar3D SDK:

Sample	Description
Countermeasure	Sample flare countermeasure which can redirect guided missiles. See Weapon Systems Overview for more information on configuring weapons within Prepar3D.
Missile	Sample missile with guidance system. See Weapon Systems Overview for more information on configuring weapons within Prepar3D.
SimpleAirplane	Basic airplane implementation. Fully functional with major systems implemented including variants with weapons and countermeasures. Can be built upon to create complex aircraft.
SimpleCar	Basic ground vehicle implementation. Fully functional with major systems implemented including AI waypoint following. Can be built upon to create complex ground vehicles.
SimpleHelicopter	Basic helicopter implementation.
WheeledTank	More advanced ground vehicle implementation of an M1128. Fully functional with major systems implemented including variants with cannons and machine guns. Can be built upon to create complex ground vehicles.

Running the PDK API Samples

The following steps show how to compile and run the PDK API samples. Each of the samples follow the same process. You must have **Visual Studio 2019** to compile the provided samples.

1. Choose the sample you wish to run and open the **SDK Samples.sln** located in the SDK directory.
2. In Visual Studio, select the **Solution Configuration (Debug/Release)** you desire to compile against.
3. From the menu bar, select **Build -> Build Solution**. You should receive a notification that the build succeeded.
4. The built sample's folder can be found in:

`Output*Solution Platform**Solution Configuration*`

5. Copy the built sample's folder directly to:

`%USERPROFILE%\Documents\Prepar3D v5 Add-ons`

. For more information, the [Distributing Add-On Packages](#) article has details on the add-on package installation process.

6. Start Prepar3D. You should be prompted to enable the add-on. Select **Yes**.
7. Similar to standard SimObjects, your **ISimObject** should be selectable from the **Select Vehicle** screen and the controls should respond to the values set in the **Control Mapping XML (DefaultControlMap.XML)** file.

NOTE: If the sample does not appear in the **Select Vehicle** screen ensure the friendly category name is added to the User Objects list found in the [Main] section of the Prepar3D.cfg. See [Understanding the PDK API Samples](#) below for more information.

Understanding the PDK API Samples

Each of the samples follow the same basic source and file structure. They can be executed on their own to learn the PDK API and ISimObject systems or they can be used as templates to create new PDK and ISimObject implementations.

All sample code is well documented and it is strongly recommended that the code of each sample be directly analyzed to learn more. For example, the *SimpleCar* sample is a very basic implementation of a ground vehicle. The following are steps to use the *SimpleCar* as a template and to understand the source and file structure of the samples.

The SimpleCar sample can be found at:

`SDK\PDK ISimObject Samples\SimpleCar`

- **Sample Folder**
 - Project (.vcxproj), Project Filters (.vcxproj.filters), and PropertyDebugging.xml

-
- For more information on Property Debugging, see [ISimObject API Debugging](#)
 - **Source**
 - Main source files for the ISimObject sample.
 - [SimObject API Overview](#)
 - [SimObject API References](#)
 - **Content**
 - **SimObject/Car** folder that contains the 3D model, panels, and other standard SimObject data.
 - [SimObject API Content Creation](#)
 - [Aircraft Configuration Files](#)
 - add-on.xml
 - [Add-On Packages](#)
 - **Output**
 - Output of the compiled PDK sample.
 - [Running the PDK API Samples](#)

Using a SimObject API Sample as a Template

After reviewing [Understanding the PDK API Samples](#) you are ready to create your own ISimObject. It is very common to use one of the samples as a template to create your own ISimObject. It is recommended that you choose the closest sample to the product you are developing. This section covers using *SimpleCar* as the starting project, but all other samples follow similar steps when using them as templated starting points. The following changes should be made to *SimpleCar*:

```
SDK\PDK ISimObject Samples\SimpleCar
```

- **Sample Folder**
 - Open the sample's Project (SimpleCar.vcxproj, but not the .vcxproj Filters file) file in a text editor. Make note of the <ProjectGuid> and change it to a new GUID.
 - Open the SDK Sample Solution (.sln) file in a text editor. Update all of the Solution's referenced project GUIDs from the previous step with the new GUID (including the GlobalSection).
 - If the project or source files are renamed, ensure to rename all instances throughout the project files (.sln, .vcxproj, and vcxproj.filters).
- **Source**
 - In SimpleCar.cpp, update the class GUID (uuid) for the SimpleCar class to a new unique value:

```
class __declspec(uuid("{3F3B2498-7E88-420B-A46D-C9EEF6BB59D0}")) SimpleCar
```

NOTE: For other samples the class may be in a header file.

- In DllMain.cpp, the RegisterSimulationCategory uses a friendly *category name* to enable selection of the object. In Prepar3D.cfg, under the [Main] section, be sure to add your object's *category name*.
 - For the *SimpleCar*, this is "GroundVehicle", thus "User Objects = Airplane, ..., GroundVehicle"
 - See [RegisterSimulationCategory](#) for more information.

- **Content**
 - Update the add-on.xml with any name/file changes.
 - SimObjects\SimpleCar\sim.cfg
 - Change the *title* to something unique for your ISimObject
 - Update the *CategoryId* to correspond to the new class GUID (uuid), which was just changed in SimpleCar.cpp
 - Create a unique *ControlsMapName* key
 - SimObjects\SimpleCar\Controls\DefaultControlMap.xml
 - Update DefaultControlMap.xml with the new mapping name from *ControlsMapName* by updating the right portion of all *Name* keys to match the changed *ControlMapName* from the sim.cfg. If you renamed the DefaultControlMap.xml file in the sim.cfg, update the *Filenname* key to match the new name.

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

Targeting Pod Sample

Contents

- [About the Targeting Pod PDK Sample](#)
- [Setting Up the Project](#)
- [Installing the Targeting Pod PDK Example](#)
- [Troubleshooting and Limitations](#)

Related Links

- [SDK Overview](#)
- [PDK Overview](#)
- [Camera Configuration](#)
- [Panel Configuration Files](#)

Overview

This sample shows how to develop a targeting pod utilizing the controllable camera for identifying and engaging multiple moving and fixed targets in air-to-air and air-to-ground engagements. It shows how to develop a controllable view with continuous stabilized surveillance of objects. The sample shows ways to shift between natural world, IR, and NVG sensor modes as well as camera controls including rotation, zoom, and locking to entities and ground locations.

The source code for the sample can be found at:

<Prepar3D SDK Path>\PDK General Samples\TargetingPodSample

About the Targeting Pod PDK Sample

The Targeting Pod PDK Sample is a simple project that demonstrates the Controllable Camera capabilities available through the PDK and using Scaleform to render the overlay. In this sample the Targeting Pod is built as a standalone dll. The Targeting Pod can also be controlled inside vehicle code, xml, or Scaleform.

The Targeting Pod PDK Sample consists of the following files:

- add-on.xml: Add-on file used to install the Targeting Pod
- DLLMain.cpp: This file contains the source code for the Targeting Pod.
- ScaleformPanelEmbeddedTargetingPod.fla: Simplified sample Scaleform fla file demonstrating how to embed the Targeting Pod in an existing Scaleform implementation.
- ScaleformPanelEmbeddedTargetingPod.swf: Compiled Scaleform swf for use in the panel.
- TargetingPod.as: Scaleform source code showing implementation of the Targeting Pod overlay. This is based off the Fury 1500 sensor camera overlay.
- TargetingPod.fla: Scaleform source code showing implementation of the Targeting Pod overlay. This is based off the Fury 1500 sensor camera overlay.
- TargetingPod.png: Used by the embedded sample as a target for the camera texture
- TargetingPod.swf: Compiled Scaleform swf for use in the panel
- TargetingPod.vcxproj: Project file
- TargetingPod.vcxproj.filters: Project filters file
- TargetingPodCallback.def: Export the functions DLLStart and DLLStop

Setting Up the Project

To build the project:

1. Open the SDK Samples.sln found in the SDK at <Prepar3D SDK Path>\ with Visual Studio 2019.
2. Right-click on Solution. Select Build to begin building the Sample.

NOTE: Paths denoted with < > represent paths on your machine that may vary based on your installation. Please ensure you have input these paths correctly, as failure to do so may cause the build to fail.

Installing the Targeting Pod PDK Example

This section discusses installation of the Targeting Pod PDK Sample. This section assumes that you have built the project and that it has generated a valid TargetingPod.dll file.

1. Copy the TargetingPod folder in the output directory to the "%USERPROFILE%\Documents\Prepar3D v5 Add-ons" directory.
2. Configure controls to operate the camera. If different control mappings are required then code changes in DLLMain.cpp must be made. By default the following will need to be mapped:
 - Camera Rotation (left/right) - Hotas Slew (left/right axis)
 - Camera Rotation (up/down) - Hotas Slew (up/down axis)
 - Toggle Sensor Mode - Hotas generic key (A0)
 - Lock target or ground location - Hotas generic key (A1)
 - Zoom in camera - Hotas generic key (A2)
 - Zoom out camera - Hotas generic key (A3)
3. Configure the reticle overlay. This sample offers three different solutions depending on the desired functionality as well as how the vehicle's panels are configured.

Standalone View

- Copy the TargetingPod.swf file to the vehicle's panel directory.
- Create a new vehicle camera definition in the vehicle's aircraft.cfg or sim.cfg. Refer to the [Camera Configuration](#) section for more information on creating camera definitions.

The following entries are required to link the reticle to the view:

- Title = "TargetingPodView"
- ScaleformOverlay=*PATH TO SWF FILE*\TargetingPod.swf

These are optional entries which may be desired depending on the training scenario:

- ShowLensFlare=TRUE
- RenderDesignators=TRUE
- ShowWeather=FALSE

Adding to the Virtual Cockpit

Note that the overlay in this sample is built using Scaleform. This can still be positioned similar to a gauge using panel coordinates. XML can be used to create a similar overlay if needed.

- Copy the TargetingPod.swf file to the vehicle's panel directory.

-
- Create a new vehicle camera definition in the vehicle's aircraft.cfg or sim.cfg. Refer to the [Camera Configuration](#) section for more information on creating camera definitions.

The following entries are required to link the reticle to the view:

- Title = "TargetingPodView"
- ScaleformOverlay=*PATH TO SWF FILE*\TargetingPod.swf
- RenderToTexture=TRUE

These are optional entries which may be desired depending on the training scenario:

- ShowLensFlare=TRUE
- RenderDesignators=TRUE
- ShowWeather=FALSE

- Add a texture entry for the TargetingPodView in the Vcockpit in the panel.cfg with the desired panel location and size values. Refer to the [Panel Configuration Files](#) section for more information.
 - texture00=TargetingPodView, 0, 0, 1024, 1024

Embedding into an existing Scaleform Panel

This method highlights how to embed an overlay into an existing swf. This sample only has a simplified overlay containing a crosshair and is not meant to function as a standalone implementation. Refer to the TargetingPod.fla and TargetingPod.as files for example code on constructing a more interactive reticle.

- If you wish to view the sample in a vehicle copy the ScaleformPanelEmbeddedTargetingPod.swf file to the vehicle's panel directory.
- To embed the Targeting Pod view into an existing Scaleform file refer to the ScaleformPanelEmbeddedTargetingPod.fla. The TargetingPod.png object is used by Prepar3D as a render target. This is done by setting the Linkage field to the same name as the camera, in this case "TargetingPodView". The texture size of the RenderToTexture view must match the original image size of this file.
- Create a new vehicle camera definition in the vehicle's aircraft.cfg or sim.cfg. Refer to the [Camera Configuration](#) section for more information on creating camera definitions.

The following entries are required to link the reticle to the view. The texture size entries must match the size of the image file used in Scaleform, in this case 1024x1024.

- Title = "TargetingPodView"
- RenderToTexture=TRUE
- TextureSizeX=1024
- TextureSizeY=1024

These are optional entries which may be desired depending on the training scenario:

- ShowLensFlare=TRUE
- RenderDesignators=TRUE
- ShowWeather=FALSE

- Add the swf to the SCALEFORM_FILE entry and add the TargetingPodView RTT camera as SCALEFORM_RTT_ELEMENT_00 in the Vcockpit in the panel.cfg. Note that if an existing SCALEFORM_FILE is already being used then only the SCALEFORM_RTT_ELEMENT_00 entry is necessary. Refer to the ScaleformPanelEmbeddedTargetingPod.fla file on how to link the TargetingPodView into a subsection of the swf. Refer to the [Panel Configuration Files](#) section for more information.
 - SCALEFORM_FILE=ScaleformPanelEmbeddedTargetingPod.swf
 - SCALEFORM_RTT_ELEMENT_00=TargetingPodView

4. Run Prepar3D and verify the camera is available to open or displaying inside the Virtual Cockpit. If not, please refer to the [Troubleshooting & Limitations](#) section below.
-

Troubleshooting & Limitations

- Check for issues with the add-on.xml file. Learn more about the add-on.xml file [here](#).
- Verify the DLL path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify all files are copied to the correct locations. If different locations are necessary than all references to these files must have the correct paths.
- If a new project is created, make sure to create a .DEF file and map it to Visual Studio.

[- top -](#)

Radar Panel Callback Sample

Contents

- [About the Radar Panel Callback PDK Sample](#)
- [Setting Up the Project](#)
- [Installing the Radar Panel Callback PDK Example](#)
- [Troubleshooting and Limitations](#)

Related Links

- [SDK Overview](#)
- [PDK Overview](#)
- [Camera Configuration](#)
- [Panel Configuration Files](#)

Overview

This sample shows how to develop a configurable radar panel controlled through an xml gauge. Attributes that can be configured include range, sweep rate, and azimuth.

The source code for the sample can be found at:

<Prepar3D SDK Path>\PDK General Samples\RadarPanel\CallbackSample

About the Radar Panel Callback PDK Sample

The Radar Panel Callback PDK Sample is a simple project that demonstrates the radar capabilities available through the PDK as well as using XML gauges to control the system. In this sample the Radar Panel is built as a standalone dll. The Radar Panel can also be controlled inside vehicle code, xml, or Scaleform.

The Radar Panel Callback PDK Sample consists of the following files:

- add-on.xml: Add-on file used to install the RadarPanel.dll and RadarExample.xml files
- RadarPanelCallback.cpp: This file contains the source code for the Radar Panel.
- RadarExample.xml: The xml gauge used to control the radar system. This needs to be referenced in a vehicle's panel.cfg file to function.
- RadarPanelCallback.vcxproj: Project file
- RadarPanelCallback.vcxproj.filters: Project filters file
- RadarPanelCallback.def: Export the functions DLLStart and DLLStop

Setting Up the Project

To build the project:

1. Open the SDK Samples.sln found in the SDK at *<Prepar3D SDK Path>* with Visual Studio 2019.
2. Right-click on Solution. Select Build to begin building the Sample.

NOTE: Paths denoted with < > represent paths on your machine that may vary based on your installation. Please ensure you have input these paths correctly, as failure to do so may cause the build

to fail.

Installing the Radar Panel Callback PDK Example

This section discusses installation of the Radar Panel Callback PDK Sample. This section assumes that you have built the project and that it has generated a valid RadarPanelCallback.dll file.

1. Copy the RadarPanelCallback folder in the output directory to the "%USERPROFILE%\Documents\Prepar3D v5 Add-ons" directory.
2. Add a panel entry to your aircraft's panel.cfg, i.e.

```
[Window Titles]
Window02=Radar

[Window02]
size_mm=265,310
Background_color=1,1,1
visible=0
position=8
windowsize_ratio=0.4
gauge00=P3DRadarExample!RadarExample, 5, 5, 255, 305
```

Load up the aircraft and select Radar from the instrument panels menu.

Troubleshooting & Limitations

- Check for issues with the add-on.xml file. Learn more about the add-on.xml file [here](#).
- Verify the DLL path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify the gauge directory path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify all files are copied to the correct locations. If different locations are necessary than all references to these files must have the correct paths.
- If a new project is created, make sure to create a .DEF file and map it to Visual Studio.

[- top -](#)

Custom Lights Sample

Overview

The **Custom Lights** sample demonstrates how to use the [PDK API](#) to create a [PdkPlugin](#) and use its [OnCustomRender](#) to draw lights. The sample leverages the power of the [PDK Services](#) to more easily access the various [Service Providers](#) available in the PDK API.

Opening the Custom Lights Project Sample

This sample is part of a series of **Custom Object Rendering Samples** and can be found at:

<Prepar3D SDK Path>\PDK General Samples\CustomLights

The easiest way to open the project is to open the **SDK Samples.sln** solution file:

<Prepar3D SDK Path>\SDK Samples.sln

Inspecting the Custom Lights Project Sample's Properties

Before running the sample or modifying it, the project properties will be inspected in order to understand how to correctly include the **PDK** library.

Select **Custom Lights** in the **Solution Explorer** and use the context menu to select the **Properties** option.

- First, set the **Configuration** combo box to **All Configurations**
- Navigate to the following section **Configuration Properties->General**:
 - Verify that the project's Configuration Type is an **Dynamic Library (.dll)**
- Navigate to the following section **Configuration Properties -> C/C++ -> General**:
 - Verify that there is an added **Additional Include Directories** path that contains the directory with the necessary PDK interfaces (e.g. **IRenderingPluginSystem.h**, **PDKServices.h**) header files:
...\\..\\..\\inc\\PDK\\\$(AdditionalIncludeDirectories)
- Note that the **Custom Lights** library will be loaded by Prepar3D and there is no library dependencies needed.
- Finally, press the **Cancel** button since the purpose of this exercise is merely to inspect the properties and not to change them

Building the Custom Lights Sample

To build the **Custom Lights**, follow these steps:

- Compile the **Custom Lights** project by right-clicking the **Custom Lights** project in the **Solution**

Explorer and selecting the **Build** option from the context menu

Installing the Custom Lights Sample

After building, navigate to the Output directory and inspect the **CustomLights** folder:

```
<Prepar3D SDK Path>\PDK General Samples\CustomLights\Output\CustomLights  
|_ add-on.xml  
|_ CustomLights.dll  
|_ ...
```

Copy the **Custom Lights** folder to the Prepar3D Add-ons directory located at:

```
%USERPROFILE%\Documents\Prepar3D v5
```

Add-ons

For more information about add-ons, visit the [Add-ons](#) article.

Running the Custom Lights Sample

Upon starting up Prepar3D there will be a prompt specifying whether the **Custom Lights** add-on should be loaded. Select **Yes**.

NOTE: If Prepar3D was opened, it should be closed at this time and reopened.

After the simulation loads, it should be apparent if the custom lights have loaded. Note the following:

- When in an outside view, such as **Locked Spot**, there is a 3x3 grid of lights attached to the user's vehicle that changes color over time.
- There is a grid of blues lights laid out around the user which stay fixed to the ground. Every second, the grid cycles between showing a part of the grid as green.

To view add-ons that are currently loaded in Prepar3D, use the menu bar and navigate to **Options->Add-ons....** In this window, the **Custom Lights** can be enabled or disabled. If it is not in this window, then either the add-on could not be found or failed to load. Retrace the steps above and make sure to visit the [Add-ons Overview](#) to learn more about the add-on process.

Understanding the Custom Lights Sample

Initializing PDK Services

Although using the [PDK Services](#) is not required, it greatly eases working with many of the [Service Providers](#). To register for the PDK Services, simply call this function before creating any [PdkPlugins](#):

```
PdkServices::Init(pPdk);
```

where **pPdk** is a pointer to the **IPdk** interface.

This would typically be done in the **DLLStart** function.

Creating the Custom Lights Plugin

Once registered, PDK Plugins can be created. In this sample, a Custom Lights Plugin is made which derives from [PdkPlugin](#). It is instantiated as follows:

```
s_pCustomLightsPlugin = new CustomLightsPlugin();
```

The PdkPlugin base class does the heavy lifting of registering for services including, but not limited to, [OnCustomRender](#) and [OnOneHz](#). The **CustomLightsPlugin** class overrides these methods and provides its own implementation which will be called by the system when those events happen.

```
virtual void OnCustomRender(IP3DCallbackParams* pParams) override
```

```
{
```

```
...
```

```
}
```

```
virtual void OnOneHz(IP3DCallbackParams* pParams) override
```

```
{
```

```
...
```

```
}
```

Rendering Custom Lights

Custom Lights can be rendered by an instance of the [IObjectRender](#) class which can be obtained by querying the [IP3DCallbackParams](#) that are available each time the [OnCustomRender](#) is called. This can be done as follows:

```
ComPtr<IObjectRenderer> spRenderService = NULL;  
if (SUCCEEDED(pParams->GetServiceProvider()->QueryService(IID_IObjectRenderer,  
IID_IObjectRenderer, (void**)&spRenderService)))  
{  
...
```

```
}
```

```
...
```

Lights can be rendered by creating groups of lights using the following method calls:

```
pRenderer->BeginLightGroup(...); // begin a new light group
```

```
...
```

```
pRenderer->AddLight(..); // add a new light, can call multiple times
```

```
...
```

```
pRenderer->EndLightGroup(...); // end the current light group  
where pRenderer is a pointer to the IObjectRender class.
```

Deinitializing PDK Services

Once all of the PDK Plugins have been destroyed, it is necessary to shut down the PDK Services to successfully unregister from all of the services it provides. This can be done by calling:

```
PdkServices::Shutdown();
```

This would typically be done in the **DLLStop** function.

[- top -](#)

Texture and Effect Render Plug-in Sample

Contents

- [About the Custom Texture and Effect Example](#)
- [Setting Up the Project](#)
- [Installing the Gauge and Effect](#)
- [Extending the Example](#)
- [Troubleshooting and Limitations](#)

Related Links

- [SDK Overview](#)
- [PDK Overview](#)
- [Programmable Gauges](#)
- [Panel Configuration Files](#)
- [Creating XML Gauges](#)

Overview

The Custom Texture and Effect Sample is a simple implementation of the new Prepar3D Custom Texture and Effect plug-in capabilities.

The source code for the sample can be found at:

<Prepar3D SDK Path>\PDK General Samples\TextureandEffectRender

Custom texture placed in a panel in the Mooney Bravo Glass Cockpit



Custom effect sample over Mooney Bravo glass cockpit view



About the Custom Texture and Effect Sample

Prepar3D v1 had a feature called DirectX Gauge which allowed for C++ gauges to render into a panel texture using DirectX 9. DirectX gauges have been deprecated for Prepar3D v4, but similar capabilities are supported via the new Texture and Effect plug-ins. Now, DirectX 11 rendering into a texture or onto the output of 3D view in Prepar3D is possible. This sample illustrates how these two plug-ins work and how Texture plug-ins can be used in combination with programmable gauges to achieve hardware accelerated rendering of gauge textures. The advantage of separating the Texture and Gauge system is that the plug-in can be a stand-alone add-on. This method provides a texture that artist can reference from any model or as a scriptable element in XML gauges.

Custom Texture and Effect Sample consists of the following files:

- [DLLMain.cpp](#): This file defines the start and end point for this DLL.
- [D3D11SampleInlineRenderer.h](#): Class declaration for the sample renderer. This class contains the callback to render contents onto the texture.
- [D3D11SampleInlineRenderer.cpp](#): Class definition for the sample renderer.
- [add-on.xml](#): Add-on file used to install the Targeting Pod.
- [Example.fx](#): Shader definition.

Additionally, a sample configuration file is provided for reference.

- [panel.cfg](#): A sample panel configuration that uses the Custom Texture in a gauge within the Mooney Bravo's glass cockpit. Note the updates to the VCockpit1 entry:
texture00=CursorTexture, 0,0,765,500.
- [camera.cfg](#): A sample camera configuration that uses the custom effect created by the sample project. Note the updates to *Virtual Cockpit* entry *PostProcess00 = CursorEffect*

NOTE: This sample is a lightweight implementation of the capabilities available to rendering plug-ins. It is expected that most of the actual DirectX drawing/rendering/animation in the example can be done in a more efficient manner, but this example should serve to demonstrate the basics of how to use the new plug-ins. For those interested in more detailed DirectX tutorials, they are available in Microsoft's DirectX Software Development Kit (SDK).

Setting Up the Project

To build the project:

1. Open the SDK Samples.sln found in the SDK at <Prepar3D SDK Path>\ with Visual Studio 2019.
2. Right-click on the main Project within the Solution Explorer and select "Properties".
3. In the Project Property Pages, select the VC++ Directories Entry.
4. Update the Include Directories path to include the paths to:
 - Pdk.h and RS_IPPluginSystem.h
 - <Prepar3D SDK Path>\inc\PDK\
5. Click the Apply button to save the settings.
6. Right-click on TextureandEffectRender in the Solution Explorer. Select Build to begin building the Sample.

NOTE: Paths denoted with < > signs represent paths on your machine that may vary based on your installation. Please ensure you have input these paths correctly, as failure to do so may cause the build to fail.

Installing the Gauge and Effect

This section discusses installation of Custom Texture and Effect Sample into an aircraft in Prepar3D. This section assumes that you have successfully built the project and that it has generated a valid a TextureandEffectRender.dll file.

1. First, copy the Copy the TextureandEffectRender folder in the output directory to the "%USERPROFILE%\Prepar3D v5 Add-ons" directory.
%APPDATA%\Lockheed Martin\Prepar3D v5
2. Next, identify the aircraft that will be used to display the gauge. Glass panel aircraft tend to provide larger surfaces on which to test; however other cockpits may also be used. Open the plane's panel.cfg file (found within <Prepar3D Path>\SimObjects\Airplanes\<Aircraft Selected>)
3. Inside the panel.cfg, identify the virtual cockpit to include the DirectX gauge on. Edit the panel.cfg by adding the **Texture** entry similar to the example shown above.
4. Add the folder path, which contains the *Example.fx* shader file, to the ShadersHLSL.cfg file located in %PROGRAMDATA%\Lockheed Martin\Prepar3D v5. Please see the [Add-ons Overview](#) article for more information.
5. Once the panel.cfg file for the aircraft has been updated appropriately, run Prepar3D and verify the gauge appears within the cockpit.

Extending the Example

The example demonstrates two simple concepts

1. Displaying a rendered texture on the Virtual Cockpit.
2. Overlaying a texture on top of a current view.

Extending to other concepts

1. Add mouse handler to the texture: A mouse handler can be added to a gauge and placed on top of a rendered texture to capture mouse events. More information can be found in the [Programmable Gauges](#) section.
2. Rendered texture can also be displayed on 2D panels.
3. XML Gauge has the capability to display Texture within an Element Object.

Troubleshooting & Limitations

-
- Check for issues with the add-on.xml file. Learn more about the add-on.xml file [here](#).
 - Verify the DLL path inside add-on.xml is valid. The path accepts both relative and absolute paths.
 - Verify all files are copied to the correct locations. If different locations are necessary than all references to these files must have correct paths
 - If a new project is created, make sure to create a .DEF file and map it to Visual Studio.

[- top -](#)

PREPAR3D

DLLMain.cpp

```
///-
/// Copyright 2015 Lockheed Martin Corporation
/// Lockheed Martin Proprietary Information
///
/// Author: Prepar3D Dev.
/// Description: This Example will render a cursor into Prepar3D as a new
///              Texture or as an Effect.
///
///-
///-
/// README:
/// SETUP Prepar3D Configuration
/// This example required additional setup in Prepar3D configuration
/// [dll.xml]
/// [panel.CFG]
/// [Camera.CFG]
///-
///-
#include <atlcomcli.h>

#include "Pdk.h"           // From Prepar3D SDK
#include "RS_IPluginSystem.h" // Main Plugin Interface
#include "D3D11SampleInlineRenderer.h"

#define     RTT_WIDTH      200
#define     RTT_HEIGHT     200

using namespace P3D;

void __stdcall DLLStart( void )
{
    CComPtr<D3D11SampleInlineRenderer> spCursorEffect = NULL;    // Callback to render an effect
    CComPtr<D3D11SampleInlineRenderer> spCursorTexture = NULL;   // Callback to render to a new texture

    CComPtr<IPdk>                     spP3dSdk = NULL;          // The PDK main interface
    CComPtr<IPluginSystem>             spPluginSystem = NULL;    // The Rendering Plugin System
    CComPtr<ITextureRenderPlugin>       spRttPlugin = NULL;       // The Plugin to create a new texture
    CComPtr<IEffectsRenderPlugin>       spEffectPlugin = NULL;    // The Plugin to create a new effect

    HRESULT hr = E_FAIL;

    // Grab the PDK reference
    if (FAILED(GetPdk(IID_IPdk, (void**)&spP3dSdk)))
    {
        return;
    }

    // Grab the Rendering System Plugin Service
    if (FAILED(spP3dSdk->QueryService( SID_IPluginSystem,
                                         IID_IPluginSystem,
                                         (void**)&spPluginSystem)))
    {
        return;
    }

    // Create a new texture for Prepar3D, this texture simply draw a Cursor on a texture
    spPluginSystem->GetOrCreatePlugin( IID_ITextureRenderPlugin, (void**)&spRttPlugin );
    spCursorTexture.Attach(new D3D11SampleInlineRenderer( false ));

    spRttPlugin->CreateTexture( "CursorTexture", RTT_WIDTH, RTT_HEIGHT, spCursorTexture );

    // Create a new Effect in Prepar3D, this draw a cursor on top of a view
    spPluginSystem->GetOrCreatePlugin( IID_IEffectsRenderPlugin, (void**)&spEffectPlugin );
    spCursorEffect.Attach( new D3D11SampleInlineRenderer( true ) );
    spEffectPlugin->CreateEffect( "CursorEffect", spCursorEffect );

    return;
}

void __stdcall DLLStop( void )
{
    //Clean up statics on module unload as necessary
}
```

- top -

PREPAR3D

D3D11SampleInlineRenderer.h

```
///-
/// Copyright 2015 Lockheed Martin Corporation
/// Lockheed Martin Proprietary Information
///
/// Author: Prepar3D Dev.
/// Description: This Example will render a cursor into Prepar3D as a new
///              Texture or as an Effect.
///              **Borrowed and Refactored from Microsoft DX11 Example
///
///

#ifndef D3D11SAMPLEINLINERENDERER_H
#define D3D11SAMPLEINLINERENDERER_H

#pragma once

#include "RS_IPluginSystem.h"
#include "IUnknownHelper.h"

#include <d3d11.h>
#include <d3dx11.h>
#include <d3dcompiler.h>
#include <xnamath.h>

class D3D11SampleInlineRenderer : public ICustomRenderCallback
{
public:

    D3D11SampleInlineRenderer( bool );
    // Callback interface from Prepar3D
    virtual void Render( IUnknown* );
    // Free resources
    void CleanupDevice( void );

    // Declare implementation for IUnknown interface
    DEFAULT_IUNKNOWN_IMPL();
    DEFAULT_IUNKNOWN_QI_IMPL( ICustomRenderCallback, IID_ICustomRenderCallback );
private:
    typedef struct
    {
        XMFLOAT3 Pos;
    } SIMPLEVERTEX;

    // Compile custom shader
    HRESULT CompileShaderFromFile( LPCSTR szFileName, LPCSTR szEntryPoint, LPCSTR szShaderModel, ID3DBlob** ppBlobOut );
    // Init resources and buffers
    HRESULT InitDevice( ID3D11Device* );
    // Call render and with the RT passed in
    void Render(ID3D11RenderTargetView*, FLOAT, FLOAT);
    ID3D11Device* mDevice;
    ID3D11VertexShader* mVertexShader;
    ID3D11PixelShader* mPixelShader;
    ID3D11InputLayout* mVertexLayout;
    ID3D11Buffer* mVertexBuffer;

    bool mIsEffect;
};

#endif
```

- top -

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

```
D3D11SampleInlineRenderer.cpp
// Copyright 2015 Lockheed Martin Corporation
// Lockheed Martin Proprietary Information
//
// Author: Prepar3D Dev.
// Description: This Example will render a cursor into Prepar3D as a new
// Texture or as an Effect.
// **Borrowed and Refactored from Microsoft DX11 Example
// -----
#include "D3D11SampleInlineRenderer.h"
#include <atlcomcli.h>

ID3D11Device* mDevice;
ID3D11VertexShader* mVertexShader;
ID3D11PixelShader* mPixelShader;
ID3D11InputLayout* mVertexLayout;
ID3D11Buffer* mVertexBuffer;

D3D11SampleInlineRenderer::D3D11SampleInlineRenderer( bool isEffect ) :
mRefCount(1),
mDevice( NULL ),
mIsEffect(isEffect),
mVertexShader( NULL ),
mPixelShader( NULL ),
mVertexLayout( NULL ),
mVertexBuffer( NULL )
{
}

// -----
// Callback entry point for Prepar3D plugin.
// -----
void D3D11SampleInlineRenderer::Render( IUnknown* data )
{
    CComPtr<ID3D11Device> device = NULL;

    // Query for a ID3D11Device from Prepar3D
    HRESULT res = data->QueryInterface( __uuidof( ID3D11Device ), (void**)&device );

    if( FAILED(res) || device == NULL )
    {
        return;
    }

    // Create buffer/inputlayout and other resources
    if( FAILED(InitDevice( device ) ) )
    {
        return;
    }

    // Query for the render target
    CComPtr<ID3D11RenderTargetView> renderTarget = NULL;
    res = data->QueryInterface<ID3D11RenderTargetView>( &renderTarget );
    if( FAILED(res) || renderTarget == NULL )
    {
        return;
    }

    // Query for additional render data which contains Render Target additional info
    CComPtr<IRenderData> renderData = NULL;
    res = data->QueryInterface<IRenderData>( &renderData );

    if( FAILED(res) || renderData == NULL )
    {
        return;
    }

    Render(renderTarget, renderData->GetTextureWidth(), renderData->GetTextureHeight());
}

// -----
// Helper for compiling shaders with D3DX11
// -----
HRESULT D3D11SampleInlineRenderer::CompileShaderFromFile( LPCSTR szFileName, LPCSTR szEntryPoint, LPCSTR szShaderModel, ID3DBlob** ppBlobOut )
{
    HRESULT hr = S_OK;

    DWORD dwShaderFlags = D3DCOMPILE_ENABLE_STRICTNESS;
#if defined( DEBUG ) || defined( _DEBUG )
    // Set the D3DCOMPILE_DEBUG flag to embed debug information in the shaders.
    // Setting this flag improves the shader debugging experience, but still allows
    // the shaders to be optimized and to run exactly the way they will run in
    // the release configuration of this program.
    dwShaderFlags |= D3DCOMPILE_DEBUG;
#endif

    ID3DBlob* pErrorBlob;
    hr = D3DX11CompileFromFile( szFileName, NULL, NULL, szEntryPoint, szShaderModel,
                                dwShaderFlags, 0, NULL, ppBlobOut, &pErrorBlob, NULL );
    if( FAILED(hr) )
    {
        if( pErrorBlob != NULL )
            OutputDebugStringA( (char*)pErrorBlob->GetBufferPointer() );
        if( pErrorBlob ) pErrorBlob->Release();
        return hr;
    }
    if( pErrorBlob ) pErrorBlob->Release();

    return S_OK;
}

// -----
// Create Direct3D device and swap chain
```

```

//-----
HRESULT D3D11SampleInlineRendererer::InitDevice( ID3D11Device* device )
{
    HRESULT hr = S_OK;

    if( mDevice == device )
        return hr;

    mDevice = device;

    // Compile the vertex shader
    ID3DBlob* pVSBlob = NULL;
    hr = CompileShaderFromFile( "ShadersHLSL\\Example.fx", "VS", "vs_4_0", &pVSBlob );
    if( FAILED( hr ) )
    {
        return hr;
    }

    // Create the vertex shader
    hr = mDevice->CreateVertexShader( pVSBlob->GetBufferPointer(), pVSBlob->GetBufferSize(), NULL, &mVertexShader );
    if( FAILED( hr ) )
    {
        pVSBlob->Release();
        return hr;
    }

    // Define the input layout
    D3D11_INPUT_ELEMENT_DESC layout[] =
    {
        { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    };
    UINT numElements = ARRAYSIZE( layout );

    // Create the input layout
    hr = mDevice->CreateInputLayout( layout, numElements, pVSBlob->GetBufferPointer(),
                                    pVSBlob->GetBufferSize(), &mVertexLayout );
    pVSBlob->Release();
    if( FAILED( hr ) )
        return hr;

    // Compile the pixel shader
    ID3DBlob* pPSBlob = NULL;
    hr = CompileShaderFromFile( "ShadersHLSL\\Example.fx", "PS", "ps_4_0", &pPSBlob );
    if( FAILED( hr ) )
    {
        return hr;
    }

    // Create the pixel shader
    hr = mDevice->CreatePixelShader( pPSBlob->GetBufferPointer(), pPSBlob->GetBufferSize(), NULL, &mPixelShader );
    pPSBlob->Release();
    if( FAILED( hr ) )
        return hr;

    // Create vertex buffer
    SIMPLEVERTEX vertices[] =
    {
        XMFLOAT3( 0.0f, 0.5f, 0.5f ),
        XMFLOAT3( 0.0f, 0.1f, 0.5f ),
        XMFLOAT3( 0.5f, 0.0f, 0.5f ),
        XMFLOAT3( 0.1f, 0.0f, 0.5f ),
        XMFLOAT3( -0.5f, 0.0f, 0.5f ),
        XMFLOAT3( -0.1f, 0.0f, 0.5f ),
        XMFLOAT3( 0.0f, -0.5f, 0.5f ),
        XMFLOAT3( 0.0f, -0.1f, 0.5f ),
    };

    D3D11_BUFFER_DESC bd;
    ZeroMemory( &bd, sizeof(bd) );
    bd.Usage = D3D11_USAGE_DEFAULT;
    bd.ByteWidth = sizeof( SIMPLEVERTEX ) * 8;
    bd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
    bd.CPUAccessFlags = 0;
    D3D11_SUBRESOURCE_DATA initData;
    ZeroMemory( &initData, sizeof(initData) );
    initData.pSysMem = vertices;
    hr = mDevice->CreateBuffer( &bd, &initData, &mVertexBuffer );
    if( FAILED( hr ) )
        return hr;

    return S_OK;
}

void D3D11SampleInlineRendererer::CleanupDevice()
{
    if( mVertexBuffer ) mVertexBuffer->Release();
    if( mVertexLayout ) mVertexLayout->Release();
    if( mVertexShader ) mVertexShader->Release();
    if( mPixelShader ) mPixelShader->Release();
}

//-----
// Render a frame
//-----
void D3D11SampleInlineRendererer::Render( ID3D11RenderTargetView* pTargetView, FLOAT width, FLOAT height )
{
    CComPtr<ID3D11DeviceContext> pDeviceContext = NULL;
    mDevice->GetImmediateContext( &pDeviceContext );

    if(pDeviceContext == NULL)
        return;

    // Setup the viewport
    D3D11_VIEWPORT vp;
    vp.Width = width;
    vp.Height = height;
    vp.MinDepth = 0.0f;
    vp.MaxDepth = 1.0f;
    vp.TopLeftX = 0;
    vp.TopLeftY = 0;

    pDeviceContext->RSSetViewports( 1, &vp );

    static float ClearColor[4] = { 0.0f, 0.0f, 0.5f, 1.0f }; // red,green,blue,alpha

    if(mIsEffect)
    {
        // Do anything specific for effect callback
    }
    else

```

```
{  
    // This is a texture type so we want to clear out the RT and render everything  
    pDeviceContext->ClearRenderTargetView( pTargetView,ClearColor );  
}  
pDeviceContext->OMSetRenderTargets( 1, &pTargetView, NULL );  
  
pDeviceContext->IASetInputLayout( mVertexLayout );  
  
// Set vertex buffer  
UINT stride = sizeof( SIMPLEVERTEX );  
UINT offset = 0;  
pDeviceContext->IASetVertexBuffers( 0, 1, &mVertexBuffer, &stride, &offset );  
  
// Set primitive topology  
pDeviceContext->IASetPrimitiveTopology( D3D11_PRIMITIVE_TOPOLOGY_LINELIST );  
  
// Render a triangle  
pDeviceContext->VSSetShader( mVertexShader, NULL, 0 );  
pDeviceContext->GSSetShader( NULL, NULL, 0 );  
pDeviceContext->PSSetShader( mPixelShader, NULL, 0 );  
  
pDeviceContext->Draw( 8, 0 );  
  
pDeviceContext->VSSetShader( NULL, NULL, 0 );  
pDeviceContext->GSSetShader( NULL, NULL, 0 );  
pDeviceContext->PSSetShader( NULL, NULL, 0 );  
  
pDeviceContext->OMSetRenderTargets( 0, NULL, NULL );  
}
```

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.
The names of the actual companies and products mentioned
herein may be the trademarks of their respective owners.

PREPAR3D

Example.fx

```
///-
-----  
/// Copyright 2015 Lockheed Martin Corporation  
/// Lockheed Martin Proprietary Information  
///  
/// Author: Prepar3D Dev.  
/// Description: Example shader file.  
///             **Borrowed and Refactored from Microsoft DX11 Example  
///  
///-
```

```
//-----  
// Vertex Shader  
//-----  
float4 VS( float4 Pos : POSITION ) : SV_POSITION  
{  
    return Pos;  
}  
  
//-----  
// Pixel Shader  
//-----  
float4 PS( float4 Pos : SV_POSITION ) : SV_Target  
{  
    return float4( 1.0f, 1.0f, 0.0f, 1.0f );      // Yellow, with Alpha = 1  
}
```

- top -

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

panel.cfg

```
// Panel Configuration file
// Mooney Bravo w/ G1000
// Copyright (c) 2001-2006 Microsoft Corporation. All rights reserved.

[Window_titles]
Window00=Main Panel
Window01=Radio
Window02=MFD
Window03=PFD
Window04=Landing Panel
Window05=Backup Gauges
Window06=ECU
Window07=Switches
Window08=Overhead Panel
Window09=Mini Panel

[Window00]
file_1024=mooney_g1000_background.bmp
file_1024_night=mooney_g1000_background_night.bmp
size_mm=1024
position=7
visible=1
no_luminous=1
ident=MAIN_PANEL
zorder=0
gauge00=G1000!G1000_PFD, 9,276,729,488
gauge01=G1000!audio_panel, 744,276, 95,488
gauge02=G1000!MFD_Mooney_LeftSide, 846,276,190,489
gauge03=SimIcons!Kneeboard Icon, 16, 540, 20, 20
gauge04=SimIcons!ATC Icon, 36, 540, 20, 20
gauge05=SimIcons!Map Icon, 56, 540, 20, 20
gauge06=SimIcons!GPS Icon, 16, 560, 20, 20
gauge07=SimIcons1024!Landing Icon, 36, 560, 20, 20
gauge08=n_number_plaque!n_number_plaque, 875,228
windowsize_ratio=1.000
window_pos=0.0,0.0
window_size=1.000,1.000

[Window01]
BACKGROUND_COLOR=2,2,2
size_mm=156,48
position=8
visible=0
ident=RADIO_STACK_PANEL
zorder=1
windowsize_ratio=1.60
window_pos=0.756,0.258
gauge00=Bendix_King_Radio!Bendix-King Radio AP,0,0,156,48

[Window02]
background_color=0,0,0
size_mm=510, 337
visible=0
position=8
ident=GPS_PANEL
zorder=2
gauge00=G1000!MFD_Mooney, 0,0,510,337

[Window03]
background_color=0,0,0
size_mm=510, 337
visible=0
position=6
ident=500
zorder=3
gauge00=G1000!G1000_PFD, 0,0,510,337

[Window04]
file_1024=mooney_g1000_landing_background.bmp
file_1024_night=mooney_g1000_landing_background_night.bmp
size_mm=1024
position=7
visible=0
no_luminous=1
ident=MAIN_PANEL_ALT2
zorder=0
view_window_rect=0,0,8192,3296
gauge00=G1000!G1000_PFD, 9,454,729,488
gauge01=G1000!audio_panel, 744,454, 95,488
gauge02=G1000!MFD_Mooney_LeftSide, 846,454,190,489
gauge03=SimIcons!Kneeboard Icon, 16, 718, 20, 20
gauge04=SimIcons!ATC Icon, 36, 718, 20, 20
```

```

gauge05=SimIcons!Map Icon, 56, 718, 20, 20
gauge06=SimIcons!GPS Icon, 16, 738, 20, 20
gauge07=SimIcons1024!VFR Icon, 36, 738, 20, 20
window_size_ratio=1.000
window_pos=0,0,0
window_size=1.000,1.000

[Window05]
file_1024=popup_backup_gauges_background.bmp
file_1024_night=popup_backup_gauges_background_night.bmp
size_mm=181,496
ident=IDENT_MISC_POPUP_1
no_luminous=1
visible=0
position=0
zorder=4
gauge00=Mooney_Bravo!Airspeed, 13, 10, 158, 158
gauge01=Mooney_Bravo!Altitude, 9, 172, 166, 157
gauge02=Mooney_Bravo!Altimeter, 13, 333, 158, 158

[Window06]
file_1024=ecu_background.bmp
file_1024_night=ecu_background_night.bmp
size_mm=259, 68
ident=IDENT_THROTTLE_PANEL
no_luminous=1
visible=0
position=6
zorder=5
gauge00=Mooney_Bravo!Thrust Controls, 0, 0, 259, 68

[Window07]
file_1024=popup_lower_panel_background.bmp
file_1024_night=popup_lower_panel_background_night.bmp
size_mm=763, 222
ident=IDENT_ELECTRICAL_PANEL
no_luminous=1
visible=0
position=8
zorder=6
gauge00=Mooney_Bravo!Master Alt Bat, 26, 24, 46, 72
gauge01=Mooney_Bravo!Avionics Switch, 84, 24, 46, 72
gauge02=Mooney_Bravo!Standby Vac, 165, 24, 46, 72
gauge03=Mooney_Bravo!Prop DeIce, 216, 24, 46, 72
gauge04=Mooney_Bravo!Pitot Heat, 267, 24, 46, 72
gauge05=Mooney_Bravo!Boost Pump, 318, 24, 46, 72
gauge06=Mooney_Bravo!Cowl Flap Switch, 372, 24, 46, 72
gauge07=Mooney_Bravo!Elevator Trim Switch, 448, 24, 46, 72
gauge08=Mooney_Bravo!Rudder Trim Switch, 510, 37, 72, 46
gauge09=Mooney_Bravo!Gear Lever, 662, 61, 38, 95
gauge10=Mooney_Bravo!Gear Light, 663, 165, 34, 18
gauge11=Mooney_Bravo!Magneto, 17, 115, 98, 98
gauge12=Mooney_Bravo!Speed Brake, 155, 146, 39, 28
gauge13=Mooney_Bravo!Master Caution, 207, 148, 39, 25
gauge14=Mooney_Bravo!Nav GPS Switch, 299, 144, 34, 38
gauge15=Mooney_Bravo!Flaps Switch, 357, 134, 41, 59
gauge16=Mooney_Bravo!Fuel Selector, 510, 124, 70, 70

[Window08]
file=upper_1024.bmp
size_mm=178
position=2
visible=0
window_size_ratio=1.6
no_luminous=1
ident=OVERHEAD_PANEL

gauge00=Mooney_Bravo!Rotating Beacon, 9,0
gauge01=Mooney_Bravo!Taxi Lights, 37,0
gauge02=Mooney_Bravo!Strobe Lights, 66,0
gauge03=Mooney_Bravo!Recognition Lights, 94,0
gauge04=Mooney_Bravo!Landing Lights, 122,0
gauge05=Mooney_Bravo!Navigation Lights, 150,0

[Window09]
position=7
size_mm=631,100
child_3d=1
background_color=0,0,0
ident=MINIPANEL

gauge00=Mooney_Bravo!Airspeed, 0,6, 81,78
gauge01=Mooney_Bravo!Turn Coordinator, 90, 6, 78,76
gauge02=Mooney_Bravo!Altitude, 180, 0, 93,87
gauge03=Mooney_Bravo!HSI, 283,0, 91,85
gauge04=Mooney_Bravo!Altimeter, 386,5, 73,76
gauge05=Mooney_Bravo!Vertical Speed, 470,6,77,77

[VCockpit01]
file=Mooney_Panel_G1000_Decals_Gray.bmp
size_mm=1024,1024
pixel_size=1024,1024
texture=$Mooney_G1000
background_color=0,0,0

gauge00=G1000!MFD_Mooney, 0,0,765,500
gauge01=G1000!audio_panel, 779,513,97,511

```

```
gauge02=G1000!G1000_PFD, 0,514,765,500
texture00=CursorTexture, 0,0,765,500 // Draw the RTT CursorTexture at this location

[vCockpit02]
file=Mooney_Panel_G1000_Decals.bmp
size_mm=512,512
pixel_size=512,512
texture=$Mooney_G1000_2
background_color=0,0,0

gauge00=Mooney_Bravo!Airspeed, 187,0,158,158
gauge01=Mooney_Bravo!Attitude, 0,177,186,177
gauge02=Mooney_Bravo!Altimeter, 0,354,158,158
gauge03=Mooney_Bravo!Annunciator, 295,301,215,47
gauge04=Mooney_Bravo!Clock, 159,468,78,44
gauge05=Mooney_Bravo!Whiskey Compass, 238,468,77,44
gauge06=Mooney_Bravo!Speed Brake, 319,480,39,28
gauge07=Mooney_Bravo!Gear Light, 159,445,34,18
gauge08=n_number_plaque!n_number_plaque, 4, 145, 89, 25

[Default View]
X=0
Y=0
SIZE_X=8192
SIZE_Y=2000

[Views]
VIEW_FORWARD_DIR= 3.0, 0.0, 0.0

[Color]
Day=255,255,255
Night=255,255,255
Luminous=201,64,64
```

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

Cameras.cfg

```
[CameraDefinition.001]
title = Cockpit
Guid = {B1386D92-4782-4682-A137-738E25D1BAB5}
Description = This is the description of the cockpit view
Origin = Cockpit
ShowPanel = Yes
SnapPbhAdjust = Ordinal
SnapPbhReturn = True
PanPbhAdjust = Ordinal
PanPbhReturn = True
Track = None
ShowAxis = FrontOnly
AllowZoom = TRUE
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = TRUE
ShowLensFlare=FALSE
Category = Cockpit
HotKeySelect=2

[CameraDefinition.002]
title = Virtual Cockpit
Guid = {C95EAB58-9E4A-4E2A-A34C-D8D9D948F078}
Description = This is the description of the virtual cockpit view.
Origin = Virtual Cockpit
MomentumEffect = Yes
SnapPbhAdjust = Swivel
SnapPbhReturn = False
PanPbhAdjust = Swivel
PanPbhReturn = False
Track = None
ShowAxis = YES
AllowZoom = TRUE
InitialZoom = 0.7
SmoothZoomTime = 2.0
ZoomPanScalar = 1.0
ShowWeather = Yes
XyzAdjust = TRUE
ShowLensFlare=FALSE
Category = Cockpit
PitchPanRate=30
HeadingPanRate=75
PanAcceleratorTime=0
HotKeySelect=1
PostProcess00 = CursorEffect // Draw the cursor effect over this camera

[CameraDefinition.003]
title = Spot
Guid = {BCA3FDD1-FB83-4BBA-8407-4922A7F0D00C}
Description = This is the description of the spot view.
Origin = Center
SnapPbhAdjust = Ordinal
SnapPbhReturn = False
PanPbhAdjust = Swivel
PanPbhReturn = False
Track = FlatChase
ShowAxis = No
AllowZoom = Yes
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = Yes
ShowLensFlare=TRUE
Category = Outside
ClipMode = Spot
PitchPanRate=30
HeadingPanRate=75
PanAcceleratorTime=0

[CameraDefinition.004]
title = Locked Spot
Guid = {BCA3FDD1-FB83-4BBA-8407-4922A7F0D00D}
Description = This is the description of the spot view.
Origin = Center
SnapPbhAdjust = Swivel
SnapPbhReturn = False
PanPbhAdjust = Swivel
PanPbhReturn = False
Track = FlatChaseLocked
ShowAxis = No
AllowZoom = Yes
```

```
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = Yes
ShowLensFlare=TRUE
Category = Outside
ClipMode = Spot
PitchPanRate=30
HeadingPanRate=75
PanAcceleratorTime=0
HotKeySelect=3

[CameraDefinition.005]
title = FlyBy
Guid = {6B79DD49-9B4A-439D-BF40-ACBF157B0BA0}
Description = This is the description of the fly by view.
Origin = Center
SnapPbhAdjust = Swivel
SnapPbhReturn = False
PanPbhAdjust = Swivel
PanPbhReturn = False
Track = FlyBy
ShowAxis = No
AllowZoom = Yes
InitialZoom = 10.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = No
ChaseDistance = 500
ChaseHeading = 15
ChaseAltitude = 10
ChaseTime = 20
ShowLensFlare=False
Category = Outside
ClipMode = Tower

[CameraDefinition.006]
title = Top-Down
Guid = {A2849229-938A-448f-8AC6-01EF2291C171}
Description = This is the description of the map or top down view.
Origin = Center
SnapPbhAdjust = Orthogonal
PanPbhAdjust = Orthogonal
Track = None
ShowAxis = Yes
AllowZoom = Yes
InitialZoom = 256
SmoothZoomTime = 2.0
ShowWeather = No
XyzAdjust = FALSE
Transition = No
ShowLensFlare=False
Category = Outside
HotKeySelect=4

[CameraDefinition.007]
title = Nearest Tower
Guid = {60BC0819-BD04-4AF6-8954-8FC8AA3545FF}
Description = This is the description of the tower view.
Origin = Tower
SnapPbhAdjust = Swivel
SnapPbhReturn = False
PanPbhAdjust = Swivel
PanPbhReturn = False
Track = Track
ShowAxis = No
AllowZoom = Yes
InitialZoom = 8.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = No
ShowLensFlare=False
Category = Tower
ClipMode = Tower
NoSorttitle = True

[CameraDefinition.008]
title = Facilities Tower
Guid = {AA8C80C0-9EE2-4284-A1C2-B20CD3F5F3D9}
Description = This is the description of the tower view.
Origin = Fixed
InstancedBased = Yes
SnapPbhAdjust = None
PanPbhAdjust = None
Track = Track
ShowAxis = No
AllowZoom = Yes
InitialZoom = 8.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = No
ShowLensFlare=False
Category = Tower
CycleHideRadius = 7
```

```
ClipMode = Tower

[CameraDefinition.009]
title = Facilities Runway
Guid = {607C4520-CA6F-4135-AE10-8BF28838068F}
Description = This is the description of the runway view.
Origin = Virtual Cockpit
InstancedBased = Yes
SnapPbhAdjust = None
PanPbhAdjust = None
TargetCategory = Fixed
Track = TrackBank
ShowAxis = Yes
AllowZoom = Yes
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = No
ShowLensFlare=FALSE
Category = Runway
CycleHideRadius = 7
ClipMode = Tower
CycleHidden=Yes

[CameraDefinition.010]
title = AI Planes
Guid = {75A8357E-AB58-4294-9416-90C73FAFDD90}
Description = This is the description of the AI aircraft view.
Origin = Center
SnapPbhAdjust = Swivel
SnapPbhReturn = False
PanPbhAdjust = Swivel
PanPbhReturn = False
Track = FlatChaseLocked
InstancedBased = Yes
ShowAxis = No
AllowZoom = Yes
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = No
ShowLensFlare=TRUE
Category = AirTraffic
TargetCategory = Container
ClipMode = Spot
PitchPanRate=30
HeadingPanRate=75
PanAcceleratorTime=0
CycleHidden=Yes

[CameraDefinition.011]
title = Multiplayer Planes
Guid = {2559BCED-9F13-4bc0-88C8-3996B9311681}
Description = This is the description of the Multiplayer other aircraft view.
Origin = Virtual Cockpit
InstancedBased = Yes
SnapPbhAdjust = None
PanPbhAdjust = None
TargetCategory = Container
Track = TrackBank
ShowAxis = Yes
AllowZoom = Yes
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = No
ShowLensFlare=FALSE
Category = MultiPlayer
CycleHideRadius = 7
ClipMode = Tower
CycleHidden=Yes

[CameraDefinition.012]
title = IR
Guid = {57A3758A-D596-4260-8AF0-302A5E4DF5CE}
SensorMode = IR
Description = Basic Infrared Sensor Camera
Origin = Cockpit
ShowPanel = No
SnapPbhAdjust = Ordinal
SnapPbhReturn = True
PanPbhAdjust = Ordinal
PanPbhReturn = True
Track = None
ShowAxis = FrontOnly
AllowZoom = TRUE
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = TRUE
ShowLensFlare=FALSE
Category = Sensor
CycleHidden=Yes

[CameraDefinition.013]
```

```
title = NVG
Guid = {61C3B6FB-FF9E-4B9C-978E-AA7C7F70F5A9}
Description = Green Colorization based NVG Sensor Camera
PostProcess00 = WhiteHotGreenColorizer
Origin = Cockpit
ShowPanel = No
SnapPbhAdjust = Ordinal
SnapPbhReturn = True
PanPbhAdjust = Ordinal
PanPbhReturn = True
Track = None
ShowAxis = FrontOnly
AllowZoom = TRUE
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = TRUE
ShowLensFlare=FALSE
Category = Sensor
CycleHidden=Yes

[CameraDefinition.014]
title = Weapons
Guid = {218F9779-FDCB-4607-9CCD-D662F8B1EB57}
Description = This is the description of the weapon view.
Origin = Center
SnapPbhAdjust = Swivel
SnapPbhReturn = False
PanPbhAdjust = Swivel
PanPbhReturn = False
Track = FlatChaseLocked
InstancedBased = Yes
ShowAxis = No
AllowZoom = Yes
InitialZoom = 1.0
SmoothZoomTime = 2.0
ShowWeather = Yes
XyzAdjust = FALSE
Transition = No
ShowLensFlare=TRUE
Category = Weapon
TargetCategory = Container
ClipMode = Spot
PitchPanRate=30
HeadingPanRate=75
PanAcceleratorTime=0
CycleHidden=Yes
```

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

Picking Sample

Contents

- [About the Camera Picking Sample](#)
- [Setting Up the Project](#)
- [Installing the Camera Picking Example](#)
- [Troubleshooting and Limitations](#)

Related Links

- [SDK Overview](#)
- [PDK Overview](#)
- [Rendering PDK Overview](#)

Overview

The Camera Picking sample demonstrates how to access the Camera System within Prepar3D, do custom 3D picking, and world ray interrogation. It also illustrates visualize the picking results using DirectX11 rendering, and how to draw text on screen.

The source code for the sample can be found at:

<Prepar3D SDK Path>\PDK General Samples\MousePicking

About the Camera Picking Sample

The Camera Picking Sample is a somewhat advanced example that demonstrates camera picking, custom rendering of simple shapes and text, accessing tool-tips from pick results, and a form of ray tracing called World Ray Interrogation. This example demonstrates how to make picking requests on a window similar to those used by the core input system to map mouse input into the virtual cockpit. The result of these pick requests is then used to determine if an interactive element was hit and how far away from the camera it is. This hit data is then visualized using custom D3D11 rendering in an effect plug-in. The camera's view and projection matrix are used to correlate the custom rendering with the cockpit. Outside the cockpit, this sample will draw the Latitude, Longitude, and Altitude of the point of the terrain under the mouse cursor.

Setting Up the Project

To build the project:

1. Open the SDK Samples.sln found in the SDK at *<Prepar3D SDK Path>* with Visual Studio 2019.
2. The *<Prepar3D SDK Path>\Inc\PDK* folder should already be set up in the project settings using relative paths. If this project folder is moved outside the SDK structure, the settings must be updated accordingly.
3. To see an example of drawing tool-tips on the custom pick results, change the 'draw_tooltips_on_pick_requests' variable in the PickHandler constructor to 'true'.
4. Right-click on Solution. Select Build to begin building the Sample.

NOTE: Paths denoted with < > represent paths on your machine that may vary based on your installation. Please ensure you have input these paths correctly, as failure to do so may cause the build to fail.

Installing the Camera Picking Example

This section discusses installation of the Camera Picking Sample.

1. Copy the MousePicking folder in the output directory to the "%USERPROFILE%\Prepar3D v5 Add-ons" directory.
2. Run Prepar3D and verify the add-on is created at startup. If not, please refer to the [Troubleshooting & Limitations](#) section below.

If the MousePicking add-on has been properly added to Prepar3D, the plugin can be used in Prepar3D in two ways. To see how custom pick requests interact with the virtual cockpit:

1. Switch to a virtual cockpit view
2. Right click, select Custom Camera -> Save Location
3. In the Manage Camera Views menu, enter a name such as "CockpitPickTest" in the Name field.
4. In the Camera Effects section, click on the Effects drop-down
5. Select MousePickingTest from the list
6. Click Add and then Save
7. Right click, select Custom Camera -> CockpitPickTest (or whatever name was given to the custom camera)

There should be 4 colored squares that draw whenever the point they represent passes a VC picking hit test.

If you change the variable 'draw_tooltips_on_pick_requests' to 'true' in the provided source code, you will see each pick request has a separate tool-tip that is drawn under each sphere.

To see how ray tracing can interact with terrain outside virtual cockpit:

1. Switch to an external view like Locked Spot or Top Down
2. Right click, select Custom Camera -> Save Location
3. In the Manage Camera Views menu, enter a name such as "ExteriorPickTest" in the Name field.
4. In the Camera Effects section, click on the Effects drop-down
5. Select MousePickingTest from the list
6. Click Add and then Save
7. Right click, select Custom Camera -> ExteriorPickTest (or whatever name was given to the custom camera)

Now there should be a String of text drawn on the screen next to the mouse cursor displaying the coordinates of the terrain under the mouse.

Troubleshooting & Limitations

- Check for issues with the add-on.xml file. Learn more about the add-on.xml file [here](#).
- Verify the DLL path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify all files are copied to the correct locations. If different locations are necessary than all references to these files must have correct paths
- If a new project is created, make sure to create a .DEF file and map it to Visual Studio.

[- top -](#)

Linux Simulation Integration Sample

Contents

- [About the Linux Simulation Integration Sample](#)
- [Setting Up the Project](#)
- [Installing the Linux Simulation Integration Sample](#)
- [Troubleshooting and Limitations](#)

Related Links

- [SDK Overview](#)
- [PDK Overview](#)
- [SimObject API Overview](#)

Overview

This is a sample on how to send UDP data back and forth from Prepar3D to a Linux Simulation program. This can allow using Prepar3D as an out the window while the Linux is used to drive the simulation (i.e. position, articulated parts, etc.).

The source code for the sample can be found at:

<Prepar3D SDK Path>\PDK General Samples\LinuxSimulationIntegration

About the Linux Simulation Integration PDK Sample

The Linux Simulation Integration PDK Sample is a simple project that demonstrates the Prepar3D's ability to integrate and communicate with Linux-based simulation engines.

Setting Up the Project

To build the project:

1. Open the SDK Samples.sln found in the SDK at *<Prepar3D SDK Path>* with Visual Studio 2019.
2. Right-click on Solution. Select Build to begin building the Sample.

NOTE: Paths denoted with < > represent paths on your machine that may vary based on your installation. Please ensure you have input these paths correctly, as failure to do so may cause the build to fail.

Installing the Linux Simulation Integration PDK Example

This section discusses installation of the Linux Simulation Integration PDK Sample. This section assumes that you have built the project and that it has generated a valid output files.

- **Prepar3D Setup**

1. This add-on will automatically try to connect to the local host and port 23332. The port can

be changed in clientServerData.h by changing "#define INTERFACE_PORT 23332"

2. Copy the LinuxSimulationIntegration folder in the output directory to the "%USERPROFILE%\Documents\Prepar3D v5 Add-ons" directory.
3. Select the Linux.Simulation.Integration from the Select Vehicle screen to load the provided simple airplane ISimObject.

- **Linux Setup**

1. Copy all the contents in Source.Linux and CommonFiles to Linux machine
2. Configure LinuxSample.cfg to the IP address and port number of Prepar3D and the rate you want the Linux executable to run.
3. Open a command window in Source.Linux and type "make" to create the exe or "make -B" to rebuild
4. Type "./LinuxSample" in command window to run

- **Additional Notes**

- Files in the CommonFiles directory must be the same between the Linux and P3D machine. Anytime any of these changes it must be recopied to other machine.
- clientServer is the UDP networking code to setup both a client and server.
- clientServerData contains the packet definitions
- In clientServer.c you can turn on a debug trace to see if packets are being sent/received. Change line 39 to "static const int trace = TRUE;" and rebuild.

Troubleshooting & Limitations

- Check for issues with the add-on.xml file. Learn more about the add-on.xml file [here](#).
- Verify the DLL path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify the SimObjects directory path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify all files are copied to the correct locations on both machines. If different locations are necessary than all references to these files must have the correct paths.
- If a new project is created, make sure to create a .DEF file and map it to Visual Studio.

[- top -](#)

Camera Sample

Contents

- [About the Camera PDK Sample](#)
- [Setting Up the Project](#)
- [Installing the Camera PDK Example](#)
- [Troubleshooting and Limitations](#)

Related Links

- [SDK Overview](#)
- [PDK Overview](#)
- [Cockpit Camera SimConnect](#)

Overview

The Camera PDK sample demonstrates how to access the Camera System within Prepar3D.

The source code for the sample can be found at:

<Prepar3D SDK Path>\PDK

About the Camera PDK Sample

The Camera PDK Sample is a simple project that demonstrates the camera control capabilities available through the PDK. Similar functionality is also available in SimConnect.

The Camera PDK Sample consists of the following files:

- DLLMain.cpp: This file defines the start and end point for this DLL. This file also contains the class definition to support the camera PDK.
- add-on.xml: Add-on file used to install the Camera Sample.

Setting Up the Project

To build the project:

1. Open the SDK Samples.sln found in the SDK at *<Prepar3D SDK Path>* with Visual Studio 2019.
2. Right-click on Solution. Select Build to begin building the Sample.

NOTE: Paths denoted with < > represent paths on your machine that may vary based on your installation. Please ensure you have input these paths correctly, as failure to do so may cause the build to fail.

Installing the Camera PDK Example

This section discusses installation of the Camera PDK Sample. This section assumes that you have built the project and that it has generated a valid CameraPDK.dll file.

1. Copy the CameraPDK folder in the output directory to the "%USERPROFILE%\Prepar3D v5

Add-ons" directory.

2. Run Prepar3D and assign controls to the HOTAS Generic A0 - A3 keys. These are used by the sample to zoom and move the camera.
 3. Verify the camera can be manipulated using these keys. **Note:** PDK control is established using the name of a view. If a view is already open using the name referenced by this sample that view will be controllable. Otherwise a new view will be opened. If controls are not functioning please refer to the [Troubleshooting & Limitations](#) section below.
-

Troubleshooting & Limitations

- Check for issues with the add-on.xml file. Learn more about the add-on.xml file [here](#).
- Verify the DLL path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify all files are copied to the correct locations. If different locations are necessary than all references to these files must have correct paths
- If a new project is created, make sure to create a .DEF file and map it to Visual Studio.

[- top -](#)

PREPAR3D

Camera.def

```
EXPORTS  
    DLLStart  
    DLLStop
```

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

PREPAR3D

DLLMain.cpp

```
///-
/// Copyright 2015 Lockheed Martin Corporation
/// Lockheed Martin Proprietary Information
///
/// Author: Prepar3D Dev.
/// Description: This Example will use the Camera PDK interface to create
///               new camera view and manipulate using the PDK functionalities
///
///-
#include <atlcomcli.h>
#include <WinError.h>

#include "Pdk.h"    //From Prepar3D SDK
#include "IWindowPluginSystem.h"
#include "IUnknownHelper.h"

using namespace P3D;

///-
/// Window Update CALLBACK CLASS
/// This callback class gets invoked every time a window is updated.
///-
class WindowCallback : public IWindowCallback
{
public:
    WindowCallback() : mWidth(200), mHeight(200), mDirection(1), m_RefCount(1) {}
    virtual void FrameUpdate( IUnknown* ) {}

    DEFAULT_IUNKNOWN_IMPL()
    DEFAULT_IUNKNOWN_QI_IMPL(IWindowCallback, IID_IWindowCallback)

private:
    unsigned int mWidth;
    unsigned int mHeight;
    int mDirection;
};

/// Use the WindowPdk interface to resize the current window
void WindowCallback::FrameUpdate( IUnknown* unknown )
{
    // Query for the Window PDK interface
    CComPtr<IWindowPdk> windowPdk = NULL;
    HRESULT res = unknown->QueryInterface<IWindowPdk>(&windowPdk);

    if(SUCCEEDED(res) && windowPdk != NULL)
    {
        if(mWidth > 400)
        {
            mDirection = -1;
        }
        else if(mWidth < 100)
        {
            mDirection = 1;
        }

        mWidth += mDirection;
        // Simply update the window by calling the set size function
        windowPdk->SetSize(mWidth, mHeight);
    }
}

///-
/// Prepar3D Loaded Handler CLASS
///-
class SystemReadyCallback : public IWindowCallback
{
public:
    SystemReadyCallback(): m_RefCount(1) {}
    virtual void FrameUpdate( IUnknown* ) {}

    DEFAULT_IUNKNOWN_IMPL()
    DEFAULT_IUNKNOWN_QI_IMPL(IWindowCallback, IID_IWindowCallback)

private:
};

/// Use the Plugin Pdk to register for windows callback or create a new Window.
void SystemReadyCallback::FrameUpdate( IUnknown* unknown )
{
    // This callback indicate that Prepar3D finish loading

    CComPtr<IWindowPluginSystem>     spPluginSystem = NULL;
    CComPtr<IWindowPlugin>             spWindowPlugin = NULL;
```

```

const static char* S_NAME = "TEST PDK WINDOW";

HRESULT res =
    unknown->QueryInterface<IWindowPluginSystem>(&spPluginSystem);

if(SUCCEEDED(res) && spPluginSystem != NULL)
{
    res = spPluginSystem->GetOrCreatePlugin( IID_IWindowPlugin,
                                              (void**)&spWindowPlugin );

    if(FAILED(res) || spWindowPlugin == NULL)
        return;

    CComPtr<WindowCallback> windowCB;
    windowCB.Attach(new WindowCallback());

    // Check if the current flight already have this camera open
    if(spWindowPlugin->HasWindow(S_NAME))
    {
        // Register for window update callback
        spWindowPlugin->RegisterInternalWindow( S_NAME, windowCB );
    }
    else
    {
        // Create new window and register for callback
        spWindowPlugin->CreateInternalWindow( S_NAME,           // Windows Name
                                               100,                // Starting x
                                               100,                // Starting y
                                               200,                // Width
                                               200,                // Height
                                               windowCB);          // On update Cb
    }
}

///-----
/// Prepar3D DLL start and end entry points
///-----
void __stdcall DLLStart( void )
{
    CComPtr<IPdk>             spP3dSdk = NULL;
    CComPtr<IWindowPluginSystem> spPluginSystem = NULL;
    CComPtr<IWindowCallback>    spSystemReadyCb = NULL;

    //Get the Prepar3D SDK
    HRESULT hr = E_FAIL;

    if (FAILED(GetPdk(IID_IPdk, (void**)&spP3dSdk)))
    {
        return;
    }

    if (FAILED(spP3dSdk->QueryService( SID_IWindowPluginSystem,
                                         IID_IWindowPluginSystem,
                                         (void**)&spPluginSystem)))
    {
        return;
    }

    spSystemReadyCb.Attach(new SystemReadyCallback() );

    // Register for a call back when the flight is loaded
    spPluginSystem->RegisterFlightLoaded(spSystemReadyCb);

    return;
}

void __stdcall DLLStop( void )
{
}

```

[- top -](#)

© 2020 Lockheed Martin Corporation. All Rights Reserved.

The names of the actual companies and products mentioned herein may be the trademarks of their respective owners.

Custom PDK Objects

Contents

- [About the Custom PDK Objects Sample](#)
- [Setting Up the Project](#)
- [Installing the Custom PDK Object Example](#)
- [Understanding the Custom PDK Object Sample](#)
- [Setting up the Menu Options](#)
- [Draw User Sim Objects](#)
- [Draw Ground Object Grid](#)
- [Draw Flashlight](#)
- [Troubleshooting and Limitations](#)

Related Links

- [SDK Overview](#)
- [PDK Overview](#)
- [Rendering PDK Overview](#)

Overview

The Custom PDK Objects sample demonstrates how to add and render custom PDK Objects to Prepar3D through the top menu

The source code for the sample can be found at:

<Prepar3D SDK Path>\PDK General Samples\CustomPDKObjects

About the Custom PDK Sample

The Custom PDK Sample demonstrates 3 custom PDK features to show how functionality can be added into Prepar3D. The features are drawing shapes around or near SimObjects, drawing a grid over the ground object, and integrating a custom effect to follow the camera. The effect in this example is a flashlight you can use in the virtual cockpit.

Setting Up the Project

To build the project:

1. Open the SDK Samples.sln found in the SDK at *<Prepar3D SDK Path>* with Visual Studio 2019.
2. The *<Prepar3D SDK Path>\Inc\PDK* folder should already be set up in the project settings using relative paths. If this project folder is moved outside the SDK structure, the settings must be updated accordingly.
3. Right-click on Solution. Select Build to begin building the Sample.

NOTE: Paths denoted with *< >* represent paths on your machine that may vary based on your installation. Please ensure you have input these paths correctly, as failure to do so may cause the build to fail.

Installing the Custom PDK Objects Sample

This section discusses installation of the Custom PDK Objects Sample.

1. Copy the CustomPDKObjects folder in the output directory to the "%USERPROFILE%\Prepar3D v5 Add-ons" directory.
2. Run Prepar3D and verify the add-on is created at startup. If not, please refer to the [Troubleshooting & Limitations](#) section below.

If the Custom PDK Objects add-on has been properly added to Prepar3D, the plugin can be used in Prepar3D through the top menu titled "Custom PDK Objects". This menu contains "Draw Ground Object Grid" and "Draw User Sim Objects" and "Draw Flashlight"

Draw Ground Object Grid when toggled on will display a series of verticle red spheres in a grid pattern across the ground. A series of bigger green spheres will move from row to row making it's way across the grid

Draw User Sim Objects when toggled on will draw a series of different colored transparent 3D shapes around the user sim object.

For using the Draw Flashlight option

1. Switch to a virtual cockpit view
2. For easier viewing, change the World time settings to night time
3. Look around the cockpit and observe that the flashlight follows the camera (It is on by default)
4. Select Draw Flashlight and observe that the effect turns on and off upon selection.

Understanding the Custom PDK Object Sample

These code segments are provided to better understand what is happening in the creation of the Custom PDK Objects. The code segments come from DLLMain.cpp located in <Prepar3D SDK Path>\PDK General Samples\CustomPDKObjects

Setting up the Menu Options

The code segments below demonstrates how to create the root top menu option.

```
m_bDrawGroundObjectGrid = true;  
m_uOneSecondTick = 0;
```

```
m_spMenuTop.Attach(P3D::PdkServices::GetMenuService()->CreateMenuItem());  
m_spMenuTop->SetType(P3D::MenuTypePdk::MENU_ITEM);  
m_spMenuTop->SetText(L"Custom PDK Objects");  
P3D::PdkServices::GetMenuService()->AddItem(m_spMenuTop->GetId(), NO_PARENT, 0);  
This segment creates an individual menu item inside the Custom PDK Object drop down  
m_spMenuDrawGrid.Attach(P3D::PdkServices::GetMenuService()->CreateMenuItem());  
m_spMenuDrawGrid->SetType(P3D::MenuTypePdk::MENU_CHECK_ITEM);  
m_spMenuDrawGrid->SetChecked(true);  
m_spMenuDrawGrid->SetText(L"Draw Ground Object Grid");  
CustomPDKObjectsPlugin::MenuCallback * callback1 = new  
CustomPDKObjectsPlugin::MenuCallback(DRAW_GRID);  
m_spMenuDrawGrid->RegisterCallback(callback1);  
P3D::PdkServices::GetMenuService()->AddItem(m_spMenuDrawGrid->GetId(), m_spMenuTop->GetId(), 0);
```

The bottom of the file contains this Invoke function which handles the selection toggling of the menu option

```
void CustomPDKObjectsPlugin::MenuCallback::Invoke(P3D::IParameterListV400 * pParams)  
{  
switch (m_EventID)  
{
```

```

case DRAW_OBJS:
{
    s_pCustomObjectsPlugin->m_bDrawUserSimObjects = !s_pCustomObjectsPlugin-
    >m_bDrawUserSimObjects;
    s_pCustomObjectsPlugin->m_spMenuDrawObjs->SetChecked(s_pCustomObjectsPlugin-
    >m_bDrawUserSimObjects);
    break;
}

}

This toggle will be acknowledged in the following OnCustomRender function and then call the actual
function for your feature
virtual void OnCustomRender(IParameterListV400* pParams) override
{
// Get the Object Renderer service from the callback params
CComPtr spRenderService = NULL;
if (SUCCEEDED(pParams->GetServiceProvider()->QueryService(
SID_ObjectRenderer, IID_IObjectRendererV400, (void**)&spRenderService)))
{
if (m_bDrawUserSimObjects)
{
DrawUserSimObjects(spRenderService, pParams->GetServiceProvider());
}
}
}
}

```

Draw User Sim Objects

"DrawUserSimObjects" is the function that gets called when the **Draw User Sim Objects** menu option is checked. This function first gets the user object from within the SimObjectManager in PDKServices and collect/converts it's location. It then uses an object renderer in the "DrawSimObjects" function to draw several different types of shapes around the user object.

Draw Ground Object Grid

"DrawGroundObjectGrid" is the function that gets called when the **Draw Ground Object Grid** menu option is checked. It creates a 10x10x5 grid of vertically aligned spheres near Eglin AFB (KVPS). A body relative offset is used to create and draw the placed spheres, as well as a row that flashes in accordance with the "OneSecondTick".

Draw Flashlight

"DrawFlashlight" is the function that gets called when the **Draw Flashlight** menu option is checked. This function creates a dynamic spot light at the camera origin and directs it at the mouse cursor. The spot light is only visible from the virtual cockpit view.

Troubleshooting & Limitations

- Check for issues with the add-on.xml file. Learn more about the add-on.xml file [here](#).
- Verify the DLL path inside add-on.xml is valid. The path accepts both relative and absolute paths.
- Verify all files are copied to the correct locations. If different locations are necessary than all references to these files must have correct paths
- If a new project is created, make sure to create a .DEF file and map it to Visual Studio.

