

Persistence Status Monitor User Guide

Version 1.3

December 2023

Absolute Persistence Status Monitor User Guide — Document revision: 5.0

This document, as well as the software described in it, is confidential and contains proprietary information protected by non-disclosure agreements. No part of this document may be reproduced in any form or disclosed to any party not bound by a non-disclosure agreement without the express written consent of Absolute® Software Corporation.

Absolute Software Corporation reserves the right to revise this document and to periodically make changes in the content hereof without obligation of such revisions or changes unless required to do so by prior agreement.

Information contained herein is believed to be correct, but is provided solely for guidance in product application and not as a warranty of any kind. Absolute Software Corporation assumes no responsibility for use of this information, nor for any infringements of patents or other rights of third parties resulting from the use of this information.

Absolute Software Corporation
Suite 1400 Four Bentall Centre
1055 Dunsmuir Street
PO Box 49211
Vancouver, British Columbia
Canada V7X 1K8

©2019-2023 Absolute Software Corporation. All rights reserved. Reproduction or transmission in whole or in part, in any form, or by any means (electronic, mechanical, or otherwise) is prohibited without the prior written consent of the copyright owner. ABSOLUTE, the ABSOLUTE logo, and PERSISTENCE are registered trademarks of Absolute Software Corporation. Other names or logos mentioned herein may be the trademarks of Absolute or their respective owners.

Contents

Downloading the Persistence Status Monitor	4
Verifying the activation status of Persistence	4
System requirements	4
Running AbtPS	4
Agent error codes	6
Persistence Status error codes	6
Script samples	7
Verifying the activation status of Persistence in an application installer	12
Get DLL version	12
Get Persistence Version	13
Get Persistence Status	13
Get Persistence Firmware Version	14
Firmware Awaiting Reboot	14
Count Required Reboots	15
Using the library	16
Testing the Secure Endpoint Agent using AbtPaaSTest.dll	16
COM API	17
Registering the COM API	17
Early bound usage	17
Late bound usage	18
AbtPaaSTest interface	19
Error conditions and handling	20
C DLL API	20
Logging	21

This document describes Absolute's Persistence Status Monitor, which you can use to check the activation status of the Absolute Persistence module embedded in the firmware of a Windows device. You can also use the tool to invoke agent calls to the Absolute Monitoring Center.

Absolute's Persistence Status Monitor includes the following components:

Name	Description
AbtPS.exe	Executable to check the status of Persistence and invoke agent calls to the Absolute Monitoring Center (wrapper for AbtPersStatusReport.dll and AbtPaasTest.dll)
AbtPersStatusReport.dll	Dynamic library for checking the status of Persistence when it is integrated with an application installer
AbtPaasTest.dll	Dynamic library for invoking agent calls to the Absolute Monitoring Center

Downloading the Persistence Status Monitor

To download the tool:

1. Log in to the Secure Endpoint Console as an Administrator.
2. On the navigation bar, click  > **Utilities**.
3. Scroll to the *Persistence Status Monitor* section of the page and click **Download**.
4. Download the `AbtPS_<version>.zip` file and extract its contents to a location of your choosing on your hard drive.

Verifying the activation status of Persistence

AbtPS can be used to check if Persistence is activated on a device and determine its Persistence version.

You can also use the AbtPS to:

- invoke an agent call to the Absolute Monitoring Center or check if a call is in process
- check if a reboot is required
- find the Absolute Identifier (ESN) assigned to the device's Secure Endpoint Agent (rpcnet)

System requirements

AbtPS is a 64-bit executable that supports the following Windows operating systems:

- Windows 10
- Windows 11

Running AbtPS

To run the AbtPS executable:

1. Open a Command Prompt with administrator privileges and navigate to the location where you extracted the contents of the `AbtPS_<version>.zip` file.

2. Run the executable using the applicable switch.

```
AbtPS [-Version | -V] [-Status | -S] [-ReadyForReboot | -R]
[-RebootsRequired:Activated | -A] [-RebootsRequired:Deactivated | -D]
[-StartCall | -C] [-IsCalling | -I] [-ESN | -E] [-LastCallResult | -L]
[-CallTimes | -T] [-CancelCall | -X] [-Help]
```

Switch	Description	Return code
-Version or -V	Shows the following: - Version of the Persistence module embedded in the device's firmware If Persistence 2.x is detected, the full firmware version is shown using the following format: <product generation>.<major version>.<minor version>.<build number> (for example: 2.2.1.18). - Version of the Secure Endpoint Agent (rpcnet) installed on the device	0: No Persistence detected 1: Persistence version 1 2: Persistence version 2.x < 0 Error
-Status or -S	Shows the status of Persistence Possible values are Persistence Activated and Persistence Deactivated .	0: Persistence is not activated 1: Persistence is activated < 0 Error
-ReadyForReboot or -R	Shows if there are outstanding server messages to process Possible values are yes and no .	0: No 1: Yes < 0 Error
-RebootsRequired:Activated or -A	Shows the minimum number of reboots to activate Persistence	0, 1, or 2 < 0 Error
-RebootsRequired:Deactivated or -D	Shows the minimum number of reboots to deactivate Persistence	0, 1, or 2 < 0 Error
-StartCall or -C	Starts an agent call to the Absolute Monitoring Center To ensure that the Secure Endpoint Agent is running, add the -StartRpcnet [-N] option. For example: <pre>AbtPS -C -N</pre>	1: Success < 0 Error
-IsCalling or -I	Shows whether the device is in the process of calling the Absolute Monitoring Center Possible values are Agent is calling and Agent is not calling .	0: Not calling 1: Calling < 0 Error

Switch	Description	Return code
-ESN or -E	Displays the device's Absolute Identifier (ESN) If the identifier ends with "0000", the Secure Endpoint Agent has not yet called in to the Absolute Monitoring Center to receive its unique Absolute Identifier. To ensure that the Secure Endpoint Agent is running, add the -StartRpcnet [-N] option. For example: <code>AbtPS -ESN -StartRpcnet</code>	0: Success < 0 Error
-LastCallResult or -L	Gets the result of the last call. Possible values are Success and Failure . Calls that are in progress are also indicated.	0: Last call failed 1: Call in progress 2: Last call successful
-CallTimes or -T	Gets the next and last call times	1: Times reported successfully to console < 0 Error
-CancelCall or -X	Cancels an in-progress call	1: Call cancelled 0: Agent was not calling < 0 Error
-Help	Shows help and the version of Persistence	

Agent error codes

The following error codes may be returned by commands related to rpcnet:

Value	Description
-1	Call was not initiated by an Administrator
-2	rpcnet is not running
-3	Agent initialization failure
-4	General failure

Persistence Status error codes

The following error codes may be returned by Persistence Status related commands:

Value	Name	Description
-1	ERRORCODE_GENERIC	An error caused by unknown reason occurred

Value	Name	Description
-2	ERRORCODE_EFI_INIT	Failed to initialize the EFI object in the library
-3	ERRORCODE_READ_ABSTATUS	Failed to read the AbtStatus UEFI variable while expecting to
-4	ERRORCODE_DW_INCOMPATIBLE	AbtStatus was read but it is not compatible with any known concrete object
-5	ERRORCODE_NO_PERSISTENCE	No Persistence-compatible firmware found (for detecting its status)
-6	ERRORCODE_REBOOTCOUNT_UNKNOWN	Reboot count could not be determined
-7	ERRORCODE_DLL_MISSING	The library is missing
-8	ERRORCODE_WMI_MISSING	WMI is missing (applies to Windows PE only)
-9	ERRORCODE_NO_PERMISSION	The user is not granted the following required permissions: - Administrator privileges - Firmware access (SeSystemEnvironmentPrivilege)

Script samples

The following PowerShell scripts demonstrate how you can use the AbtPS executable.

Sample 1: Activating Persistence

This script checks if Persistence is activated, and then reboots the device if a reboot is required.

```

1  #!/usr/bin/env powershell
2  # Copyright (c) 2020 Absolute Software Corporation, All rights reserved.
3  # Reproduction or transmission in whole or in part, in any form or by any means,
4  # electronic, mechanical or otherwise, is prohibited without the prior written
5  # consent of the copyright owner.
6  #
7
8  Write-Host "Activating Persistence..."
9
10 # Force an rpcnet agent call
11 Write-Host "Starting an agent call..."
12 Start-Process -FilePath .\AbtPS.exe -Wait -ArgumentList "-StartCall" -WindowStyle
   Hidden
13
14 # Wait for the call to complete
15 do
16 {
17     Write-Host "Sleeping 2 seconds"
18
19     Start-Sleep -s 2
20
21     $process = Start-Process -FilePath .\AbtPS.exe -ArgumentList "-IsCalling" -
   PassThru -WindowStyle Hidden

```

```

22     Wait-Process -InputObject $process
23 }
24 while ($process.ExitCode -eq 1)
25
26 Write-Host "Call is complete"
27
28 # Check to see if a reboot is necessary
29 $process = Start-Process -FilePath .\AbtPS.exe -Wait -ArgumentList "-
RebootsRequired:Activated" -PassThru -WindowStyle Hidden
30
31 if ($process.ExitCode -ne 0)
32 {
33     Write-Host "Reboot is needed. Restarting the computer..."
34     Start-Sleep 2
35     Restart-Computer -Force
36 }
37
38 # Check the activation status
39 $process = Start-Process -FilePath .\AbtPS.exe -Wait -ArgumentList "-Status" -
PassThru -WindowStyle Hidden
40
41 if ($process.ExitCode -eq 1)
42 {
43     Write-Host "Persistence is activated"
44 }
45 elseif ($process.ExitCode -eq 0)
46 {
47     Write-Host "Persistence is not activated"
48 }

```

Sample 2: Install and activate Persistence

This script runs the Windows core agent installer (`AbsoluteCoreAgent.msi`), initiates activation, and then reboots the device if a reboot is required.

```

1  #!/usr/bin/env powershell
2  # Copyright (c) 2020, 2022 Absolute Software Corporation, All rights reserved.
3  # Reproduction or transmission in whole or in part, in any form or by any means,
4  # electronic, mechanical or otherwise, is prohibited without the prior written
5  # consent of the copyright owner.
6  #
7  # Force execution to be as an Administrator
8  # Info on the Requires statement:
9  # https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\_requires?view=powershell-5.1
10 # Requires -RunAsAdministrator
11 #
12 # Usage:
13 # InstallAndActivate.ps1 -package <path to AbsoluteWinCoreAgent.zip>
14 # If you wish to specify a temp folder where the agent package will be extracted to,
15 # specify that using the -temp parameter, like this:
16 # InstallAndActivate.ps1 -package <path to AbsoluteWinCoreAgent.zip> -temp
"C:\agent"
17 # If the temp path doesn't exist, an attempt will be made to create it.

```

```
18 #
19 # Get command line argument
20 param(
21     [Parameter(Mandatory=$true)]$package, # path to the agent package (zip) file
22     $temp = $PSScriptRoot # temp folder path where the agent package will be
    unzipped
23 );
24
25 $LogFilePath = (Join-Path $PSScriptRoot "InstallAndActivate.log")
26 function Log {
27     param ([string] $message)
28
29     Write-Host $message
30
31     # Write the log file
32     if (Test-Path $LogFilePath) {
33         Add-Content -Path $LogFilePath -Value $message
34     }
35     else {
36         Set-Content -Path $LogFilePath -Value $message
37     }
38 }
39
40 Log "Installing Persistence..."
41
42
43 # Check to see if the specified agent package exists
44 if (!(Test-Path $package))
45 {
46     Log ("The agent package was not found: " + $package)
47     exit -1;
48 }
49
50 # Check temp folder. Attempt to create it if it doesn't exist.
51 if (!(Test-Path $temp))
52 {
53     New-Item $temp -ItemType Directory
54 }
55
56 Log ("Extracting " + $package + " to " + $temp)
57 Expand-Archive $package $temp -Force # force the unzip assuming the new package is
    what the user wants to install
58
59 $AgentInstaller = "AbsoluteCoreAgent.msi"
60 $AgentInstallerPath = (Join-Path $temp $AgentInstaller)
61
62 # Check to see if the Agent installer file is in the current folder
63 if (!(Test-Path $AgentInstallerPath))
64 {
65     Log ($AgentInstaller + " is missing from the agent package.")
66     exit -1;
67 }
68
69 $AgentInstallerDataFile = "AbsoluteAgent.dat"
70
71 # Check to see if the Agent installer data file is in the current folder
72 if (!(Test-Path (Join-Path $temp $AgentInstallerDataFile)))
```

```

73 {
74     Log ($AgentInstallerDataFile + " is missing from the agent package.")
75     exit -1;
76 }
77
78 $AgentInstallerSignatureFile = "AbsoluteAgent.sig"
79
80 # Check to see if the Agent installer signature file is in the current folder
81 if (!(Test-Path (Join-Path $temp $AgentInstallerSignatureFile)))
82 {
83     Log ($AgentInstallerSignatureFile + " is missing from the agent package.")
84     exit -1;
85 }
86
87 $AgentInstallerLogPath = (Join-Path $temp "AbsoluteCoreAgent.log")
88
89 # Set up the arguments to msixexec.exe
90 # Run msixexec with verbose logging to help in case of errors
91 $AgentInstallerArgs = @(
92     "/i"
93     "`"$AgentInstallerPath`"
94     "/qn"
95     " /l*v"
96     "`"$AgentInstallerLogPath`"
97 )
98
99 $timestamp = Get-Date
100 Log $timestamp
101 Log ("Agent being installed by '" + $env:USERDOMAIN + "\" + $env:UserName + "'")
102
103 # Run the agent installer using msixexec and wait for it to finish
104 Log ("Starting msixexec with args: " + $AgentInstallerArgs)
105 Write-Host "This may take a while"
106 $process = (Start-Process -FilePath msixexec.exe -ArgumentList $AgentInstallerArgs -
    NoNewWindow -Wait -PassThru)
107
108 # Done. Indicate success or failure in the Event Log
109 # Only do cleanup if the agent was successfully installed. In case of an error,
110 # the logs are helpful in any diagnosis of an issue.
111 if (0 -eq $process.ExitCode) {
112     Log "msixexec returned success."
113 }
114 else {
115     Log ("msixexec returned error " + $process.ExitCode)
116     Log "Quitting"
117
118     Exit -1
119 }
120
121 Log "Checking the Persistence activation status"
122
123 $process = Start-Process -FilePath .\AbtPS.exe -Wait -ArgumentList "-Status" -
    PassThru -WindowStyle Hidden
124
125 if ($process.ExitCode -eq 1)
126 {
127     Log "Persistence is activated"

```

```
128 }
129 elseif ($process.ExitCode -eq 0)
130 {
131     Log "Persistence is not activated"
132 }
133 elseif ($process.ExitCode -eq -5)
134 {
135     Log "Persistence is not installed. Quitting."
136     Exit -2
137 }
138 else {
139     Log ("Exit code " + $process.ExitCode + " returned. Quitting.")
140     Exit -2
141 }
142
143 # Force an rpcnet agent call
144 Log "Starting an Agent call..."
145 Start-Process -FilePath .\AbtPS.exe -Wait -ArgumentList "-StartCall" -PassThru -
    WindowStyle Hidden
146
147 # Wait for the call to complete
148 do
149 {
150     Write-Host "Sleeping 2 seconds"
151
152     Start-Sleep -s 2
153
154     $process = Start-Process -FilePath .\AbtPS.exe -ArgumentList "-IsCalling" -
        PassThru -Wait -WindowStyle Hidden
155 }
156 while ($process.ExitCode -eq 1)
157
158 Log "Call is complete"
159
160 Log "Checking to see if a reboot is necessary"
161
162 $process = Start-Process -FilePath .\AbtPS.exe -Wait -ArgumentList "-
    RebootsRequired:Activated" -PassThru -WindowStyle Hidden
163
164 if ($process.ExitCode -ne 0)
165 {
166     Log "Reboot is needed. Restarting the computer..."
167     Start-Sleep 2
168     Restart-Computer -Force
169 }
170 else {
171     Log "No reboot required"
172 }
173
174 Log "Checking the Persistence activation status"
175
176 $process = Start-Process -FilePath .\AbtPS.exe -Wait -ArgumentList "-Status" -
    PassThru -WindowStyle Hidden
177
178 if ($process.ExitCode -eq 1)
179 {
180     Log "Persistence is activated"
```

```

181 }
182 elseif ($process.ExitCode -eq 0)
183 {
184     Log "Persistence is not activated"
185 }
186 elseif ($process.ExitCode -eq -5)
187 {
188     Log "Persistence is not installed"
189 }
190 else {
191     Log ("Exit code " + $process.ExitCode + " returned")
192 }

```

Verifying the activation status of Persistence in an application installer

NOTE This section applies only if you have integrated Absolute Persistence with an application installer and you want to check the status of Persistence. For more information, see the *Absolute Application Installer Integration Guide*.

You can use the following libraries to query the status of Persistence on a device when you integrate an application installer with Absolute Persistence.

- 32-bit: **AbtPersStatusReport.dll**
- 64-bit: **AbtPersStatusReport64.dll**

You can use a few different components to determine if the installation or uninstallation process has completed, and if Absolute Persistence is in the desired state.

This section describes the methods, which are C++ functions, that are exposed by the libraries.

Get DLL version

This function returns the version of the DLL as a 32-bit value in the format **A.B.C.D**, where each part of the value is a number between 0 and 255.

```
DWORD GetDllVersion(VOID);
```

Return value	Description
0xFFFFFFFF (-1)	Operation failed
else	DLL Version

To extract each part:

- A = (version >> 24) & 0xFF
- B = (version >> 16) & 0xFF

- C = (version >> 8) & 0xFF
- D = version & 0xFF

Get Persistence Version

The *Get Persistence Version* function determines the version of the Absolute Persistence module detected in the firmware.

```
int GetPersistenceVersion(VOID);
```

Return value	Description
< 0	Operation failed
0	Persistence not detected
1	Persistence 1.0 detected
2	Persistence 2.x detected

Error codes

The Get Persistence Version function may return the following [error code](#):

- Failure code: `ERRORCODE_EFI_INIT`.

Get Persistence Status

The *Get Persistence Status* function determines if Persistence 1 or Persistence 2.x is activated on the firmware.

NOTE This function applies to Persistence 2.x and some Persistence 1.0 devices.

```
int GetPersistenceStatus(VOID);
```

Return value	Description
< 0	Operation failed
0	Persistence deactivated
1	Persistence activated

Error codes

The Get Persistence Status function may return the following [error codes](#):

- Failure codes: `ERRORCODE_GENERIC`, `ERRORCODE_EFI_INIT`, `ERRORCODE_READ_ABSTATUS`, `ERRORCODE_FW_INCOMPATIBLE`, and `ERRORCODE_NO_PERSISTENCE`.

- Any error code that `GetPersistenceVersion` returns.

NOTE If a device does not support either version of Persistence, `ERRORCODE_GENERIC` is returned.

Get Persistence Firmware Version

The *Get Persistence Firmware Version* function returns the Persistence 2.x firmware version in the following format:

<ProductGeneration>.<MajorVersion>.<MinorVersion>.<BuildNumber> (for example: 2.2.1.18)

NOTE If Persistence 1 is detected, version 1.1.0.0 is always shown. This API is not able to detect Persistence 1 firmware information.

```
int GetPersistenceFirmwareVersion(int* ProductGeneration, int* MajorVersion, int*
MinorVersion, int* BuildNumber );
```

Return value	Description
< 0	Operation failed
0	No Persistence detected
1	Persistence 1 detected
2	Persistence 2.x detected
ProductGeneration	Pointer to an integer that receives the production generation of the Persistence 2.x firmware 1: Persistence 1 2: Persistence 2.x
MajorVersion	Pointer to an integer that receives the major version of the Persistence 2.x firmware
MinorVersion	Pointer to an integer that receives the minor version of the Persistence 2.x firmware
BuildNumber	Pointer to an integer that receives the build number of the Persistence 2.x firmware

Firmware Awaiting Reboot

The *Firmware Awaiting Reboot* function determines if content is in either of the firmware mailboxes, AuthBlob or Mailbox. If so, it assumes that the Absolute Monitoring Center (or any other server that the device communicated with) is waiting for an action to take effect and thus returns **yes**, indicating that a reboot is required.

NOTE This function applies to Persistence 2.x and some Persistence 1 devices.

```
int IsFirmwareAwaitingReboot(VOID);
```

Return value	Description
< 0	Operation failed
0	Reboot is not required
1	Reboot is required

Error codes

The Firmware Awaiting Reboot function may return the following [error codes](#):

- Failure codes: `ERRORCODE_GENERIC`, `ERRORCODE_EFI_INIT`, and `ERRORCODE_NO_PERSISTENCE`.
- Any error code that `GetPersistenceVersion` returns.

Count Required Reboots

If the *Firmware Awaiting Reboot* function returns **yes**, the *Count Required Reboots* function can be used to determine the number of reboots required to complete the installation or uninstallation process.

This function interrogates the `AbtStatus` UEFI variable to see if the `AUTHKEY_READY` flag is set. Depending on the flag and what the desired state should be, this function returns an integer.

```
int CountRequiredReboots(int expectedState);
```

Return value	Description
< 0	Operation failed
0	No reboot is required (the device is in the same state as the expected state)
1	One reboot is required (the <code>AUTHKEY_READY</code> is set and the device is in a state that is different from the expected state)
2	Two reboots are required (the <code>AUTHKEY_READY</code> is not set and the device is in a state that is different from the expected state)

expectedState	Description
0	Expectation that firmware is ultimately deactivated
1	Expectation that firmware is ultimately activated
else	Unsupported value

Error codes

The Count Required Reboots function may return the following [error codes](#):

- Failure code: `ERRORCODE_EFI_INIT`
- Any error code that `GetPersistenceVersion`, `GetPersistenceStatus`, or `IsFirmwareAwaitingReboot` returns.

Using the library

You can use the library as a static linked library or a dynamic linked library.

Static library

The library produces `.lib` and `.dll` files that can be used to statically link the library to a C/C++ project and consume its exposed functions.

Dynamic library

The following is an example based on Python 2.7 that can load and execute the library.

```

1  #
2  # This sample script is provided as an example of how to use the provided APIs in
   Python
3  #
4
5  import sys
6  import ctypes
7
8  # Confirm if the parameter is provided.
9  if len(sys.argv) != 2:
10     print('Please provide path to the library to run test.')
11     sys.exit()
12
13  # Load library from the provided path.
14  libPath = str(sys.argv[1])
15  print('Loading library at:', libPath)
16  dll = ctypes.CDLL(libPath)
17
18  # Invoke library methods:
19  print('-----')
20  print('DLL Version:' + str(dll.GetDllVersion()))
21  print('Persistence Version:' + str(dll.GetPersistenceVersion()))
22  print('Persistence Status:' + str(dll.GetPersistenceStatus()))
23  print('Firmware Awaiting Reboot:' + {True:'Yes', False:'No'}
   [dll.IsFirmwareAwaitingReboot()==1])
24  print('Reboot needed to be deactivated:' + str(dll.CountRequiredReboots(0)))
25  print('Reboot needed to be activated:' + str(dll.CountRequiredReboots(1)))
26
27  # Dispose the object to release library.
28  del dll
29

```

Testing the Secure Endpoint Agent using AbtPaaSTest.dll

AbtPaaSTest is a simple component that tests scenarios involving interaction with the Secure Endpoint Agent (rpcnet). You can use this component to check if an agent call is in progress, or to start a call.

The component includes two APIs:

- a COM API that exposes the functionality naturally to coding environments such as C++, C#, PowerShell, and Python
- a C DLL API that is useful for C environments and others that can call exported DLL functions, such as InstallShield and MSI

AbtPaaSTest.dll is a 64-bit executable that must be run with administrative privileges. To use AbtPaaSTest.dll with its COM interfaces, COM registration is required. The C DLL API does not require registration and can be used directly by any 64-bit process.

COM API

The COM API can be used by early-binding environments, such as C# programs, with a reference. Early binding infers that the programming environment interrogates the capabilities of the component at design time. The COM API can also be used by late binding environments such as PowerShell or Python, where the methods and properties are discovered on-the-fly at runtime.

Registering the COM API

To register the COM API, open an elevated command prompt and run the following command *as an Administrator*:

Component Registration

```
regsvr32 -i AbtPaaSTest.dll
```

You only need to run this command once.

Early bound usage

C#

In Visual Studio, add a reference to your project to use AbtPaaSTestLib. Note that the project needs to be an x86 (32-bit) project.

In your project, instantiate an instance of the object as follows:

IsAgentCalling

```
var abtPaaSTest = new AbsolutePaaSTest();

if (abtPaaSTest.IsAgentCalling)
    Console.WriteLine("Agent is calling");
else
    Console.WriteLine("Agent is not calling");
```

StartAgentCall

```
var abtPaaSTest = new AbsolutePaaSTest();
abtPaaSTest.StartAgentCall();
```

C++

In C++ using ATL:

IsAgentCalling

```
CComPtr<IAbsolutePaaSTest> pAbtPaaSTest;
pAbtPaaSTest.CoCreateInstance(CLSID_AbsolutePaaSTest);

VARIANT_BOOL vbCalling = VARIANT_FALSE;
pAbtPaaSTest->get_IsAgentCalling(&vbCalling);
std::cout << "IsAgentCalling returned " << ((vbCalling == VARIANT_TRUE) ? "True" :
"False") << std::endl;
```

StartAgentCall

```
CComPtr<IAbsolutePaaSTest> pAbtPaaSTest;
pAbtPaaSTest.CoCreateInstance(CLSID_AbsolutePaaSTest);

pAbtPaaSTest->StartAgentCall();
```

Late bound usage

In an environment such as PowerShell or Python, create a COM object using the ProgID `Absolute.AbtPaaSTest`.

PowerShell

In an elevated 64-bit PowerShell console, use the following code:

IsAgentCalling

```
$AbtPaaSTest = New-Object -COMObject Absolute.AbtPaaSTest
if ($AbtPaaSTest.IsAgentCalling)
{
    Write-Host "Agent is calling"
}
else
{
    Write-Host "Agent is not calling"
}
```

StartAgentCall

```
$AbtPaaSTest = New-Object -COMObject Absolute.AbtPaaSTest

$AbtPaaSTest.StartAgentCall()
```

Python

In an elevated 64-bit Python environment, use the following code:

IsAgentCalling

```
import win32com.client as win32
abtPaaSTest = win32.Dispatch("Absolute.AbtPaaSTest")

if abtPaaSTest.IsAgentCalling:
    print("Agent is calling")
else:
    print("Agent is not calling")
```

StartAgentCall

```
import win32com.client as win32
abtPaaSTest = win32.Dispatch("Absolute.AbtPaaSTest")

abtPaaSTest.StartAgentCall()
```

AbtPaaSTest interface**Methods**

StartAgentCall(): starts an agent call and returns Success or Failure

Properties

IsAgentCalling: checks if an agent call is in progress and returns True or False

GetESN: returns the device's Absolute Identifier (ESN)

Error conditions and handling

The COM API reports errors through the standard COM error reporting mechanism: the *IErrorInfo* interface is exposed by the component.

High-level environments such as C# and PowerShell will automatically check for errors using this mechanism. For example, C# will throw an exception, and the exception object will include the error code and error message from the component.

C DLL API

Using the C DLL API, the *AbtPaaSTest.dll* exposes three functions: **StartAgentCall**, **IsAgentCalling** and **GetESN**.

StartAgentCall

The *StartAgentCall* function starts an agent call and returns one of the following values:

Return value	Description
1	Success
-1	Caller does not have administrative rights
-2	rpcnet is not running
-3	Failed to initialize the agent connection

IsAgentCalling

The *IsAgentCalling* function checks if an agent is in progress and returns one of the following values:

Return value	Description
1	Agent is calling
0	Agent is not calling
-1	Caller does not have administrative rights
-2	rpcnet is not running
-3	Failed to initialize the agent connection
-4	Unable to retrieve the agent busy status

GetESN

The *GetESN* function returns the device's Absolute Identifier (ESN). If the identifier ends with "0000", the Secure Endpoint Agent has not yet called in to the Absolute Monitoring Center to receive its unique Absolute Identifier.

The following parameters are supported:

- `LPWSTR szESN`: buffer to `wchar_t` array where the ESN will be placed
- `DWORD szESNSize`: size of the `szESN` buffer, expressed as a character count

NOTE The *GetESN* function supports the Unicode character set only.

Return value	Description
0	Success
-1	Unable to get ESN
-2	Unable to get ESN due to insufficient privileges (administrator privileges required)
-3	Null pointer passed to function

Logging

The *AbtPaaSTest* component logs information and error messages in the Windows event log. You can view entries from *AbtPaaSTest* in the application log.

All API calls are logged. If any errors occur, they are logged as well.