PROFILE TUNING SUITE (PTS)

EXTENDED AUTOMATING – USING PTS CONTROL API

Disclaimer and Copyright Notice

Table of Contents

1 PTS Terminology

- **IUT (Implementation Under Test)**: The device, component or subsystem to be tested.

- **Workspace**: A group of profile and protocol test suites to be tested against the Implementation Under Test. A workspace may be thought of as representing a particular device, component or subsystem.

- **Project**: A profile or protocol test suite and its associated data files. One or more projects may be present in a workspace. Each project represents a profile or protocol supported by the IUT.

- **ICS (Implementation Conformance Statement)**: Official declaration of the profile or protocol features and functions that are supported by the IUT. Each item in the ICS selects one or more tests that must be executed in order to demonstrate proper implementation.

- **IXIT (Implementation Extra Information for Testing)**: Data items, such as the *Bluetooth* Device Address (BD_ADDR), that are specific to a particular IUT. In general, IXIT items represent data that cannot be specified in advance by the programmer who created a test case or test suite.

- **ETS (Executable Test Suite)**: Each profile or protocol specified for use in *Bluetooth* wireless technology has an accompanying test specification. An ETS is a programmatic representation of the test purposes found in a particular test specification. Test cases in an ETS are executed under the control of the Profile Tuning Suite.

- **Test Purposes vs. Test Cases**: A test specification defines a number of test purposes which describe the environment that must be present to perform a test of a particular feature or function, the proper procedure to perform a test, and the expected outcome of a test.

  A test case is specific implementation of a test purpose, for example, a test case found in a PTS Executable Test Suite.

- **Test Case Naming**: Each test purpose defined in a test specification is identified by a name which is created according to a standard policy. The name identifies which groups of tests a particular test belongs to along with the nature of the test. Test purpose names are in a format similar to

TP/ABC/BV-01-I

In the PTS, the naming format is modified slightly to change the "/" and "-" characters to "_" characters. In addition, since a particular test purpose may be defined for more than one operational "role", the role for a specific test case is inserted into the name. A PTS test case name corresponding to the example test purpose above might be

TC_CLIENT_ABC_BV_01_I

("TC" replacing "TP" since PTS implements test cases not test purposes.)

2 Automating PTS

The Profile Tuning Suite (PTS) offers three features which can be used in automated testing:

1. "Operator-less Operation" allows the interactive prompts that appear during the execution of a test case to be processed by user written software which can inspect each message and take appropriate action.

   This feature can be used with either of the following program control features and is described in the "Automating – Using Implicit Send" reference document.

2. "Scripted Operation" where a set of test cases can be selected and run as a group. The group can be executed as needed, or scheduled for execution at a later time.

   The "Scripting" reference document describes this feature.

3. "Fully Automated Operation" – PTS provides an Application Programming Interface (API) which allows complete control of the software. User written programs can take advantage of the API in order to open Workspaces, select Projects, execute Test Cases, and many other functions.

   This document describes "Fully Automated Operation".

2.1 "Fully Automated Operation"

Completely unattended operation of PTS can be achieved by using the PTS Control API. There are three parts to "Fully Automated Operation":

1. PTS.exe: The main application program for the Profile Tuning Suite. It accepts calls from user written programs that use the PTS Control API and takes the appropriate action;

2. Client Application: An application program written by a PTS user that makes use of the PTS Control API in order to have PTS carry out various functions. The PTS Control API portion of a Client Application may be part of a larger program that interfaces to a test system platform that has the capability of also controlling the *Bluetooth* device that is being tested;

3. PTSControl.dll: A Windows Dynamic Link Library (DLL) that provides the PTS Control API interface and data type information to the Component Object Model (COM) manager. This information is normally registered with COM during the installation of PTS.

The PTS Control API is implemented using Microsoft's Component Object Model (COM) which means that any Windows based programming language that supports COM can be used to develop the Client Application. Some test system platforms also support the use of COM directly, eliminating the need to create a separate Client Application program.

A C/C++ header file (PTSControl.h) is provided for developers using Microsoft's Visual C or Visual C++. The information in the file can be used as a guideline for developing the proper COM interface declarations for other programming languages or COM enabled applications.

A sample Client Application – PTSControlClient – is also provided as part of the PTS installation. This application, written in Visual C++, exercises many of the functions available in the API. It can be a valuable reference when developing your own Client Applications.

The source code for PTSControlClient, the PTSControl.h file, and Visual Studio (2008) build files are located in the

> C:\ Program Files [(x86)]\Bluetooth SIG\Bluetooth PTS\SampleCode\PTSControlClient folder in the PTS installation.

3 PTS and the PTS Control API

PTS needs to be started in COM Server mode in order to allow Client Applications access to the PTS Control API. A normal execution of PTS is not in COM Server mode and will likely cause the Client Application to fail when it attempts to connect to the Server.

When PTS is started in COM Server mode a "console-like" User Interface will appear. Users should be able to monitor all the log messages that normally appear in the Log window of PTS while tests are automatically executed.

To start PTS in COM Server mode the Client Application must:

1. Initialize Component Object Model via CoInitialize() or CoInitializeEx().

2. Create an instance of the IPTSControl COM object by calling the Windows API function CoCreateInstance(). The call to CoCreateInstance() will start PTS in COM Server mode (as long as PTS is not currently running.).

4 General API usage

The following points should be kept in mind while using the PTS Control API:

- C and C++ programs should include “PTSControl.h” to get the interface definitions, error codes and other information.

- The Windows API function CoInitialize() (or CoInitializeEx()) must be called before using any functions from the API.

- All functions return an HRESULT value. Data that may be returned from a particular function will be returned via the function’s parameter list.

- All text string parameters used by the API are in the wide character (Unicode UTF-16) format. The Client Application should generally be built using wide character strings. When this is not possible, Windows API routines such as MultiByteToWideChar() and WideCharToMultiByte() should be used to convert strings being sent to or received from the API to the wide character format.

  All string parameters are assumed to “C-style”, that is, terminated with a NUL character (ASCII value 0x0000).

- There are no data values being passed over the API that are allocated in one context (such as PTS.exe) and deallocated in the other (the Client Application). Additionally, there are no C++ objects used by the interface.

  This means that it is safe to use a Debug configuration build of a Client Application with the PTS Control API. (PTS.exe, which contains the PTS Control API, is built using the Release configuration.)

5 Functions in the PTS Control API

5.1 Opening/Creating a Workspace

The first step in using the PTS Control API is to open or create a Workspace. An open workspace is needed for all of the Project and Test Case related functions in the API.

5.1.1 CreateWorkspace()

Declaration: HRESULT CreateWorkspace(ULONGLONG ullBthAddr, LPCWSTR pszPathOfPtsFile, LPCWSTR pszWorkspaceName, LPCWSTR pszWorkspacePath);

Parameters: ullBthAddr: A 64 bit unsigned integer that contains the Bluetooth Device Address (BDADDR) of the Implementation Under Test (IUT).

Note that a 64 bit value is used even though a BDADDR is only 48 bits in length. The BDADDR is located in the least significant 48 bits (six bytes) and the upper two bytes must have a value of 0x0000.

pszPathOfPtsFile: A Unicode character string that contains the path to a ICS file describing the features of the IUT that was previously exported from the Test Plan Generator (TPG)/Qualified Listing Interface (QLI).

The use of a full file path is recommended since the name is processed by the running instance of PTS. The instance of PTS may have a different current working directory than the Client Application.

pszWorkspaceName: A Unicode character string containing the name of the Workspace to be created. This is just the name of the Workspace, not a file path to the intended Workspace location.

pszWorkspacePath: A Unicode character string containing the path to the folder where the new Workspace should be created. The new Workspace will be created in a subfolder of this location, with the name of the subfolder coming from the pszWorkspaceName parameter.

The folder path must exist before making this call, PTS will not create any folders that are missing from the path specification.

The use of a full path to the folder is recommended since it is processed by the running instance of PTS. The instance of PTS may have a different current working directory than the Client Application.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

Use this function to create a new Workspace from a set of ICS values exported from the TPG/QLI. The exported file contains all of the ICS items declared for a given device, some of which may not be applicable to PTS.

A Project will be created in the Workspace for each Profile or Protocol available in the current installation of PTS. Any Profile or Protocol not available in the current installation will be ignored.
The BDADDR of the IUT may not be known at the time the workspace is created. In this case, use any convenient value and update it later using the UpdateIxitParam() function.

5.1.2 OpenWorkspace()

Declaration: HRESULT OpenWorkspace(LPCWSTR pszPathOfWorkspace);

Parameters: pszPathOfWorkspace: A Unicode character string containing the path to the Workspace to be opened. The name of the Workspace file will be "<workspace name>.pqw6" in the "root" of the Workspace folder. Providing a path to a legacy ".pqw" workspace will automatically trigger workspace conversion to the new format.

The use of a full path for the Workspace file is recommended since it is processed by the running instance of PTS. The instance of PTS may have a different current working directory than the Client Application.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

Use OpenWorkspace() to open an existing Workspace and load all of the Projects found therein.

5.2 Working with Projects

Once a Workspace has been opened, the following functions may be used to obtain information about the available Projects.

5.2.1 GetProjectCount()

Declaration: HRESULT GetProjectCount(UINT* pcProjects);

Parameters: pcProjects: A pointer to a 32 bit unsigned value that will receive the number of Projects (Test Suites) that are available in the current Workspace.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function returns the number of Projects that are available in the current Workspace. This count can be used as the upper boundary when using GetProjectName() in a loop to acquire the names of the available Projects.

5.2.2 GetProjectName()

Declaration: HRESULT GetProjectName(UINT iProject, LPWSTR* ppszProjectName);

Parameters: iProject: The zero based index to a Project in the currently open Workspace.

ppszProjectName: A pointer to a Unicode string pointer that will receive the address of the name of the selected Project.

The actual pointer should be initialized to NULL before making this call.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

GetProjectName() returns a pointer to the name of a selected Project in the current Workspace. The Project is selected by the iProject value which must be less than the value returned by GetProjectCount().

ppszProjectName is a pointer to a Unicode character string and should be initialized to NULL before calling GetProjectName(). The pointer is filled in by GetProjectName() with the address of the string containing the Project name. The contents of the string pointed at by ppszProjectName should not be modified.

Example:

```
LPWSTR pszProjectName;

pszProjectname = NULL;
<interface pointer>->GetProjectName(0, &pszProjectName);
```

Upon return from GetProjectName(), pszProjectName will point at the name of the selected Project and can be used as

```
wprintf(L”The Project name is %s\n”, pszProjectName);
```

5.2.3 GetProjectVersion()

Declaration: HRESULT GetProjectVersion(LPCWSTR pszProjectName, DWORD* pProjVersion);

Parameters: pszProjectName: A Unicode character string that contains the name of a Project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

pPTSVersion: A pointer to a 32 bit unsigned integer that receives the version number of the specified Project.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

Returns the version of a named Project (Test Suite) as a four byte value packed into a 32 bit unsigned integer. Each byte represents one piece of a standard Windows version number:

- Byte 3: Major version number
- Byte 2: Minor version number
- Byte 1: Update release number
- Byte 0: Build sequence number

For example, for a Test Suite whose version number is 7.5, update 0, build number 4 (7.5.0.4), the value returned from GetProjectVersion() would be 0x07050004.

5.3 Working with Test Cases

After a Workspace has been opened, information about the Test Cases available in a given Project can be obtained, and Test Cases may be executed using these functions.

5.3.1 GetTestCaseCount()

Declaration: HRESULT GetTestCaseCount(LPCWSTR pszProjectName, UINT* pcTestCases);

Parameters: pszProjectName: A Unicode character string that contains the name of a project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

pcProjects: A pointer to a 32 bit unsigned value that will receive the number of Test Cases that are available in the selected Project.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

This function returns the number of Test Cases that are available in the specified Project. This count can be used as the upper boundary when using GetTestCaseName() and GetTestCaseDescription() in a loop to acquire the names and descriptions of the available Test Cases.

5.3.2 GetTestCaseName()

Declaration: HRESULT GetTestCaseName(LPCWSTR pszProjectName, UINT iTestCase, LPWSTR* ppszTestCaseName);

Parameters: pszProjectName: A Unicode character string that contains the name of a Project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

iTestCase: The zero based index to a Test Case in the selected Project.

ppszTestCaseName: A pointer to a Unicode string pointer that will receive the address of the name of the selected Test Case.

The actual pointer should be initialized to NULL before making this call.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

GetTestCaseName() returns a pointer to the name of a selected Test Case in a given Project. The Test Case is selected by the iTestCase value which must be less than the value returned by GetTestCaseCount().

ppszTestCaseName is a pointer to a Unicode character string and should be initialized to NULL before calling GetTestCaseName(). The pointer is filled in by GetTestCaseName() with the address of the string containing the Test Case name. The contents of the string pointed at by ppszTestCaseName should not be modified.

Example:

```
LPCWSTR          pszProjectName = L"IOPT":
LPWSTRpszTestCaseName;

pszTestCaseName = NULL;
<interface pointer>->GetTestCaseName(pszProjectName, 0, &pszTestCaseName);
```

Upon return from GetTestCaseName(), pszTestCaseName will point at the name of the selected Test Case and can be used as

```
wprintf(L"The Test Case at index %u in Project %s is s\n",
        0, pszProjectName, pszTestCaseName);
```

5.3.3 GetTestCaseDescription()

Declaration: HRESULT GetTestCaseDescription(LPCWSTR pszProjectName, UINT iTestCase, LPWSTR* ppszTestCaseDesc);

Parameters: pszProjectName: A Unicode character string that contains the name of a Project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

iTestCase: The zero based index to a Test Case in the selected Project.

ppszTestCaseDesc: A pointer to a Unicode string pointer that will receive the address of the description of the selected Test Case.

The actual pointer should be initialized to NULL before making this call.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

GetTestCaseDescription() returns a pointer to the description of a selected Test Case in a given Project. The description is the same information that is returned by the PTS User Interface when right-clicking on a Test Case and selecting "Show Purpose". It is usually the first paragraph or so from the definition of the corresponding Test Purpose in the applicable test specification.

The Test Case is selected by the iTestCase value which must be less than the value returned by GetTestCaseCount().

ppszTestCaseDesc is a pointer to a Unicode character string and should be initialized to NULL before calling GetTestCaseDescription(). The pointer is filled in by GetTestCaseDescription() with the address of the string containing the Test Case description. The contents of the string pointed at by ppszTestCaseDesc should not be modified.

Example:

```
LPCWSTR          pszProjectName = L”IOPT”:
LPWSTRpszTestCaseDesc;

pszTestCaseName = NULL;
<interface pointer>->GetTestCaseDescription(pszProjectName, 0,
        &pszTestCaseDesc);
```

Upon return from GetTestCaseDescription(), pszTestCaseDesc will point at the name of the selected Test Case and can be used as

```
wprintf(L“The description for the Test Case at index %u in Project %s is s\n”,
        0, pszProjectName, pszTestCaseDesc);
```

5.3.4 IsActiveTestCase()

Declaration: HRESULT IsActiveTestCase(LPCWSTR pszProjectName, LPCWSTR pszTestCase, BOOL* pbIsActive);

Parameters: pszProjectName: A Unicode character string that contains the name of a Project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the “TestCaseView” window of the PTS User Interface.

pszTestCase: A Unicode character string that contains the name of a TestCase in the specified Project.

The Test Case name is case sensitive and must match the name shown in the “TestCaseView” window of the PTS User Interface.

pbIsActive: A pointer to BOOL value that will receive the state of the Test Case.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function returns TRUE via pbIsActive if the selected Test Case is active (enabled) in the specified Project. False is returned via pbIsActive if the Test Case is not active (disabled).

5.3.5 RunTestCase()

Declaration: HRESULT RunTestCase(LPCWSTR pszProjectName, LPCWSTR pszTestCase);

Parameters: pszProjectName: A Unicode character string that contains the name of a Project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

pszTestCase: A Unicode character string that contains the name of a TestCase in the specified Project that is to be executed.

The Test Case name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

RunTestCase() executes the specified Test Case. The return value is the status of the RunTestCase() function call itself and not the final verdict from the Test Case.

To get the final verdict of the Test Case, along with the other execution log information, a logging function must be connected to the PTS Control API. See section 5.5.1 ("Logging") for more information.

5.3.6 StopTestCase()

Declaration: HRESULT StopTestCase();

Parameters: None.

Return values: A value greater than or equal to zero if successful.

A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

StopTestCase() submits a request to stop the executing Test Case.

StopTestCase will wait for the Test Case to complete for a period of time set by SetPTSCallTimeout() API. If the time out duration is not specified StopTestCase will wait for 2 minutes. If the executing Test Case completes within specified period of time, the API returns S_OK. If the test case fails to complete, the API will return PTSCONTROL_E_TESTCASE_TIMEOUT. If other issues occur, the API returns PTSCONTROL_E_INTERNAL_ERROR.

5.3.7 GetTestCaseCountFromTSSFile ()

Declaration: HRESULT GetTestCaseCountFromTSSFile (LPCWSTR pszProjectName, UINT* pcTestCases);

Parameters: pszProjectName: A Unicode character string that contains the name of a project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

pcProjects: A pointer to a 32 bit unsigned value that will receive the number of Test Cases that are available in the selected Project.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function is similar to GetTestCaseCount() and returns the number of Test Cases that are available in the specified Project. The main difference is that GetTestCaseCount() relies on profile description .html files located in Documentation\Bluetooth\ETS references folder. Only the test cases described in these files will be included in the count of Test Cases returned by GetTestCaseCount() API. GetTestCaseCountFromTSSFile does not have this limitation and returns the total test case count.

GetTestCasesFromTSSFile() API is exposed by IPTSControlEx interface. A pointer or reference to that interface must be obtained from the automation object to use this API:

```
CComQIPtr<IPTSControlEx> spPTSEx = m_pPTS;
if (spPTSEx.p != NULL)
{
	UINT cTestCases = 0;
	HRESULT hr = spPTSEx->GetTestCaseCountFromTSSFile(pszProjectName, &cTestCases);
}
```

5.3.8 GetTestCasesFromTSSFile ()

Declaration: HRESULT GetTestCasesFromTSSFile (LPCWSTR pszProjectName, VARIANT *pvTestCases);

Parameters: pszProjectName: A Unicode character string that contains the name of a project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the “TestCaseView” window of the PTS User Interface.

pvTestCases: A pointer to a VARIANT object that will receive the list of all test cases found in the project specified by pszProjectName. The test case list will be packaged into a safe array type object.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function complements the GetTestCaseName() API and returns the list of Test Cases that are available in the specified Project. The main difference is that GetTestCaseName() relies on profile description .html files located in Documentation\Bluetooth\ETS references folder. Only the test cases described in these files will be returned by GetTestCaseName() API. GetTestCasesFromTSSFile does not have this limitation and returns the complete test case list.

GetTestCasesFromTSSFile () API is exposed by IPTSControlEx interface. A pointer or reference to that interface must be obtained from the automation object to use this API:

```
CComQIPtr<IPTSControlEx> spPTSEx = m_pPTS;
if (spPTSEx.p != NULL)
{
	CComVariant vt;
	HRESULT hr = spPTSEx-> GetTestCasesFromTSSFile(pszProjectName, &vt);
}
```

5.4 Working with ICS and IXIT data

5.4.1 UpdatePics()

Declaration: HRESULT UpdatePics(LPCWSTR pszProjectName, LPCWSTR pszEntryName, BOOL bValue);

Parameters: pszProjectName: A Unicode character string that contains the name of a Project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

pszEntryName: A Unicode character string that contains the name of a ICS item defined in the specified Project that is to be updated.

The ICS item name is case sensitive and must match the name shown in the ICS editor dialog of the PTS User Interface.

bValue: The new value for the ICS item.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

This function updates the value of a ICS item for the selected Project. The value may be set to TRUE or FALSE as appropriate.

5.4.2 UpdatePixitParam()

Declaration: HRESULT UpdateIxitParam(LPCWSTR pszProjectName, LPCWSTR pszParamName, LPCWSTR pszNewParamValue);

Parameters: pszProjectName: A Unicode character string that contains the name of a Project that is available in the current Workspace.

The Project name is case sensitive and must match the name shown in the "TestCaseView" window of the PTS User Interface.

pszEntryName: A Unicode character string that contains the name of a IXIT item defined in the specified Project that is to be updated.

The IXIT item name is case sensitive and must match the name shown in the IXIT editor dialog of the PTS User Interface.

pszNewParamValue: A Unicode character string that contains the new value to be assigned to the specified IXIT item.

See the table below for restrictions on the contents of this string.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

This function updates the value of a IXIT item for the selected Project. IXIT items have data types associated with them. This means that the pszNewParamValue string must only contain characters that are appropriate for the associated data type.

Data Type	**Legal Characters For The New Value**
BITSTRING	0,1
BOOLEAN	The words TRUE or FALSE (case sensitive)
IA5STRING	No restrictions
INTEGER	Decimal digits 0 to 9
OCTETSTRING	Hexadecimal “digits” 0 to 9 and A to F (“A” to “F” must be upper case)

5.5 Logging and unattended operation

During normal operation, PTS sends informational outputs to the Test Case execution log in the upper right hand corner of the PTS User Interface. Client Application programs may take over this functionality and divert the log output to a function within the application.

Additionally, at various points during the execution of a Test Case, PTS will display dialog boxes prompting the test operator to take an action. There are two ways to remove the test operator and replace them with user written code:

1. Develop a custom Implicit Send DLL as described in the "Automating – Using Implicit Send" reference document.

2. Use the functions provided in the PTS Control API to divert the operator prompts to the Client Application.

5.5.1 Logging

There are two steps necessary to divert the test case execution log to a Client Application application. The first is to create a COM based object derived from IPTSControlClientLogger. The object needs to implement the following functions that are needed by the Component Object Model:

- AddRef()
- Release()
- QueryInterface()

In addition, the object needs to implement a function called Log() which will be called every time that PTS would normally send something to the Test Case execution log window.

The second step to diverting the execution log is to provide an instance of the IPTSControlClientLogger derived object to the PTS Control API via the SetControlClientLoggerCallback() function.

5.5.1.1 IPTSControlClientLogger::Log()

Declaration: HRESULT Log(PTS_LOGTYPE logType, LPCWSTR szLogType, LPCWSTR szTime, LPCWSTR pszMessage);

Parameters: logType: The type of the logging event. See the table below.

szLogType: A Unicode character string that contains the name of the Test Case event being logged.

szTime: A Unicode character string that contains the time of the event being logged. The time is a value in milliseconds from the start of the Test Case execution.

pszMessage: A Unicode character string that contains the information about the event that is being logged.

Return values: The user implementation of this function should return a value greater than or equal to zero if successful.
The user implementation of this function should return a value less than zero if not successful.

The three character strings passed to the Log() function contain the three pieces of information that are normally shown in the Test Case execution log window in the PTS User Interface. These strings may be used to create log output that looks just like the information displayed in the Test Case execution window. For example, the following event might be displayed in the Test Case execution log window:

```
+3666 ms
Receive event:      :[3]HCI?HCI_READ_LOCAL_VERSION_INFORMATION_COMPLETE_EVENT=PDU:{
                    status:HCI_OK,
                    hciVersion:4,
                    hciRevision:5360,
                    lmpVersion:4,
                    manufacturerName:10,
                    lmpSubversion:5360
                   }
```

The corresponding character strings passed to the Log() function would be:

```
szTime
szLogType:      pszMessage
```

The szTime string starts with a blank line that is used in the User Interface to provide a visual break between logged events.

pszMessage includes leading spaces for each line after the first one. This is used to provide the indentation of the message information in the PTS User Interface.

The values for the logType parameter are found in PTSControl.h and are listed here:

logType Value	**Usage**
PTS_LOGTYPE_INFRASTRUCTURE	This type is used for log messages from the PTS Control API that normally would not appear in the Test Case Execution log.
PTS_LOGTYPE_START_TEST	The "Start Test Case" event that is logged when a Test Case begins executing.
PTS_LOGTYPE_END_TEST	The "Test Case ended" event that is logged when a Test Case has completed execution.
PTS_LOGTYPE_IMPLICIT_SEND	Not currently used in PTS. The value however is used in the PTSControlClient sample program.
PTS_LOGTYPE_ERROR	An "Error" event logged when an executing Test Case has encountered an internal problem.
PTS_LOGTYPE_SEND_EVENT	"Send event"s are used to log data being sent from the PTS to the Implementation Under Test (IUT) or to an internal component of the currently executing Test Case.
PTS_LOGTYPE_RECEIVE_EVENT	"Receive event"s are used to log data received from the Implementation Under Test (IUT) or from an internal component of the currently executing Test Case.
PTS_LOGTYPE_FINAL_VERDICT	The "Final Verdict" of the Test Case execution. The result of the Test Case execution – PASS, FAIL, etc – can be found in the pszMessage string.
PTS_LOGTYPE_PRELIMINARY_VERDICT	At various points during Test Case execution, a Test Case will issue a "Preliminary Verdict". The most negative verdict issued becomes the "Final Verdict" of the Test Case. "Preliminary Verdict" messages can be used to determine at what point during execution that a Test Case failed.
PTS_LOGTYPE_EVENT_SUMMARY	At various points during Test Case execution, a Test Case will issue a "Verdict Description" that is placed in the both the execution log and Output window in the lower left hand corner of the PTS User Interface. PTS_LOGTYPE_EVENT_SUMMARY is used to indicate those messages.
PTS_LOGTYPE_MESSAGE	This type is used for any log data that is not covered by one of the above. This is the majority of the output in the Test Case execution log.

5.5.1.2 SetControlClientLoggerCallback()

Declaration: HRESULT SetControlClientLoggercallback(IPTSControlClientLogger* pLogger);

Parameters: pLogger: A pointer to an instance of an IPTSControlClientLogger based COM object as described above.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function hooks the “logger” object to the PTS Control API. After this function has been called, Test Case execution log data will be sent to the Log() function instead of the PTS User Interface.

5.5.2 Unattended operation

Most Test Cases in PTS assume that a test operator is available to perform various actions on the IUT and to confirm that events have occurred. “Unattended operation” allows the test operator to be replaced by user written software.

The prompts to the test operator are referred to as “MMI”s in the PTS documentation. The feature that presents the MMIs to the operator is commonly referred to as “Implicit Send”.

As mentioned earlier, there are two forms of unattended or “Operator-less Operation”. One such method is to create a custom Implicit Send DLL to replace the one normally supplied with PTS. This is discussed in the "Automating - Using Implicit Send" reference document.

An alternative is to use the Implicit Send support provided by the PTS Control API. This support allows the Client Application to process the various MMIs.

There are three steps necessary to divert the various MMIs to the Client Application.

1. Create a COM based object derived from either IPTSImplicitSendCallbackEx or IPTSImplicitSendCallback.

   The IPTSImplicitSendCallback**Ex** object was added in PTS version 4.6 to address some limitations with in the IPTSImplicitSendCallback object. New Client Applications should use the IPTSImplicitSendCallback**Ex** object.

   The IPTSImplicitSendCallback object is provided for backwards compatibility with existing Client Applications, though it is highly recommended that existing Applications be upgraded to use IPTSImplicitSendCallback**Ex.**

2. Implement a callback function that PTS Control API will invoke whenever an MMI occurs.

3. Register the callback function with PTS Control API.

5.5.3 IPTSImplicitSendCallbackEx object

The IPTSImplicitSendCallback**Ex** object needs to implement the following functions that are needed by the Component Object Model:

- AddRef()
- Release()
- QueryInterface()

In addition, the object needs to implement a function called OnImplicitSend() which will be called every time that PTS would normally use Implicit Send to popup a prompt message for the test operator.

Finally, an instance of the IPTSImplicitSendCallback**Ex** derived object needs to be registered with the PTS Control API via the RegisterImplicitSendCallback**Ex**() function.

5.5.3.1 IPTSImplicitSendCallbackEx::OnImplicitSend()

Declaration: HRESULT OnImplicitSend(LPCWSTR pszProjectName, WORD wID,
LPCWSTR pszTestCase,LPCWSTR pszDescription, DWORD style,
LPWSTR pszResponse, DWORD responseSize, BOOL* pbResponseIsPresent)

Parameters: pszProjectName: A Unicode string containing the name of the Project that contains the currently executing Test Case.

wID: An unsigned 16 bit value that uniquely identifies the MMI in the Project.

pszTestCase: A Unicode character string that contains the name of the currently executing Test Case.

pszDescription: A Unicode character string that contains the prompt text that would normally be shown in a popup dialog.

style: An unsigned 32 bit value the identifies the style of the MMI. (See below)

pszResponse: A Unicode string buffer of size responseSize. The implementation of OnImplicitSend() will copy the response text to be sent to the executing test case into this buffer.

responseSize: The size of the pszReponse buffer.

pbResponseIsPresent: A pointer to a Win32 BOOL that should be set to TRUE if response text has been placed in the pszResponseBuffer, FALSE if no response text is being returned.

Return values: The user implementation of this function should return a value greater than or equal to zero if successful.
The user implementation of this function should return a value less than zero if not successful.

Every MMI used in the PTS Test Suites (Projects) contains a unique tag that identifies it. The actual text of the prompt may change over time, but the unique tag (generally) will not. Client Applications can use the tag to identify a particular MMI rather than counting on the contents of the prompt.

There are three parts to the unique tag:

1. wID: The MMI identifier.

2. pszProjectName: wID values may be used for the different MMIs in different Projects. The combination of (pszProjectName,wID) uniquely identifies the MMI for a specific project.

3. pszTestCaseName: At times, the response to a particular MMI may depend on the currently executing Test Case. pszTestCaseName allows the same MMI to be used by different Test Cases within a given Project.

The style parameter to OnImplicitSend() provides direction about the contents of the first three parameters (wID, pszProjectName, pszTestCaseName) along with an indication of the expected return value.

Style name	**Value**	**Message Type**	**Buttons Displayed by the default Implicit Send DLL**
MMI_Style_Ok_Cancel1	0x11041	Simple prompt	OK, Cancel Default: OK
MMI_Style_Ok_Cancel2*	0x11141	Simple prompt	Cancel
MMI_Style_Ok	0x11040	Simple prompt	OK
MMI_Style_Yes_No1	0x11044	Simple prompt	Yes, No Default: Yes
MMI_Style_Yes_No_Cancel1	0x11043	Simple prompt	Yes, No, Cancel Default: Yes
MMI_Style_Abort_Retry1	0x11042	Simple prompt	Abort, Retry, Ignore Default: Abort
MMI_Style_Edit1	0x12040	Request for data input	OK, Cancel Default: OK
MMI_Style_Edit2	0x12140	Select item from a list	OK, Cancel Default: OK

***Note: When ImplicitSendStyle() is called with style MMI_Style_Ok_Cancel2, implementation may signal the IUT the requested action after the message tag is identified but it should not block in the function. Otherwise, it may block PTS from progressing. Implementation should always return "OK".**

Section 3.4 of "Automating – Using Implicit Send" contains a complete description of the various styles and the expected return values. "Automating – Using Implicit Send" should be consulted for more information.

The data return mechanisms used by OnImplicitSend() differ slightly from the ImplicitSendStyle() API function described in "Automating – Using Implicit Send".

- Returning a "positive" response:
    - ImplicitSendStyle() returns a pointer to a character string.
    - For OnImplicitSend(), the character string is copied into the pszResponse buffer. No more than (responseSize – 1) characters should be copied into the buffer. (The -1 leaves room for the NUL character that must terminate the response.
        - Use of the wcscpy_s() function is recommended to make sure that no more than the maximum number of characters are copied to the buffer.

      Additionally, setting pbResponseIsPresent to TRUE will tell PTS that there is data in the pszResponseBuffer.

      For example, to return a value of "OK":

```
wcscpy_s(pszResponse,responseSize,L"OK");
```

```
*pbResponseIsPresent = TRUE;
```

- Returning a “negative” response:
    - ImplicitSendStyle() returns a NULL pointer.
    - For OnImplicitSend(), a negative response is indicated by setting pbResponseIsPresent to FALSE. When pbResponseIsPresent is FALSE, PTS will ignore the contents of the pszResponse buffer.

      For example, to return a “negative” response:

      ```
      *pbResponseIsPresent = FALSE;
      ```

5.5.3.2 RegisterImplicitSendCallbackEx()

Declaration: HRESULT RegisterImplicitSendCallback**Ex**(IPTSImplicitSendCallback**Ex*** pCallback);

Parameters: pCallback: A pointer to an instance of an IPTSImplicitSendCallback**Ex** based COM object as described above.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function hooks the Implicit Send handler object to the PTS Control API. After this function has been called, MMIs will be sent to the OnImplicitSend() function instead of the PTS User Interface.

5.5.3.3 UnregisterImplicitSendCallbackEx()

Declaration: HRESULT UnregisterImplicitSendCallback**Ex**(IPTSImplicitSendCallback**Ex*** pCallback);

Parameters: pCallback: A pointer to an instance of an IPTSImplicitSendCallback**Ex** based COM object as described above.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function disconnects the Implicit Send handler object from the PTS Control API. After this function has been called, MMIs will once again be sent to the PTS User Interface.

5.5.4 IPTSImplicitSendCallback object

*This object is provided for backwards compatibility with existing Client Applications and use of the IPTSImplicitSendCallback**Ex** object is preferred.*

The IPTSImplicitSendCallback object needs to implement the following functions that are needed by the Component Object Model.

- AddRef()
- Release()
- QueryInterface()

In addition, the object needs to implement a function called OnSend() which will be called every time that PTS would normally use Implicit Send to popup a prompt message for the test operator.

Finally, an instance of the IPTSImplicitSendCallback derived object needs registered with the PTS Control API via the RegisterImplicitSendCallback() function.

5.5.4.1 IPTSImplicitSendCallback::OnSend()

Declaration: HRESULT OnSend(LPCWSTR pszProjectName, WORD wID, LPCWSTR pszTestCase, LPCWSTR pszDescription);

Parameters: pszProjectName: A Unicode string containing the name of the Project that contains the currently executing Test Case.

wID: An unsigned 16 bit value that uniquely identifies the MMI in the Project.

pszTestCase: A Unicode character string that contains the name of the currently executing Test Case.

pszDescription: A Unicode character string that contains the prompt text that would normally be shown in a popup dialog.

Return values: The user implementation of this function should return a value greater than or equal to zero if successful.
The user implementation of this function should return a value less than zero if not successful.

The four parameters for the OnSend() function are identical to the first four parameters of the OnImplicitSend() function described above. Please refer to section 5.5.3.1 for more information about these parameters.

5.5.4.1.1 Limitations with IPTSImplicitSendCallback::OnSend()

IPTSImplicitSendCallback::OnSend() has a few limitations that have been addressed by IPTSImplicitSendCallback**Ex**::OnImplicitSend().
The Implicit Send feature uses an additional value that describes the presentation style of the MMI. The presentations styles determine if the MMI should be presented with

- OK and Cancel buttons
- Only a Cancel button
- Yes and No buttons
- A data input dialog
- A list of choices from which one item may be selected

In addition, the MMI style defines the expected return value from the Implicit Send handler.

- The test operator's choice of OK or Cancel, Yes or No.
- The test operator pressed the Cancel button for MMI styles that only include a Cancel button.
- Input typed into a dialog by the test operator.
- The item selected from the list of choices.

- The MMI style is NOT passed to OnSend(). This means that a Client Application will not be presented with any of the information listed above. The MMI style is supplied to OnImplicitSend().

- OnSend() does not provide a mechanism to return a value other than "the OK button was pressed". OnImplicitSend() supports all of the expected return values noted above.

  The return status from OnSend() should be greater than or equal to zero *if the function completes successfully.* A return status less than zero indicates to the PTS Control API that *the OnSend() function was not able to complete successfully.*

  In other words, the status value returned from OnSend() only indicates whether or not the function was successful, not a particular value that should be returned from the list of possibilities given above.

5.5.4.2 RegisterImplicitSendCallback()

Declaration: HRESULT RegisterImplicitSendCallback(IPTSImplicitSendCallback* pCallback);

Parameters: pCallback: A pointer to an instance of an IPTSImplicitSendCallback based COM object as described above.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 ("API Error Codes").

This function hooks the Implicit Send handler object to the PTS Control API. After this function has been called, MMIs will be sent to the OnSend() function instead of the PTS User Interface.

5.5.4.3 UnregisterImplicitSendCallback()

Declaration: HRESULT UnregisterImplicitSendCallback(IPTSImplicitSendCallback* pCallback);

Parameters: pCallback: A pointer to an instance of an IPTSImplicitSendCallback based COM object as described above.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function disconnects the Implicit Send handler object from the PTS Control API. After this function has been called, MMIs will once again be sent to the PTS User Interface.

5.5.5 Additional Automation APIs Exposed by IPTSControlEx Interface

All APIs described in this section are exposed by IPTSControlEx interface. A pointer or reference to that interface must be obtained from the automation object to use these APIs:

```
CComQIPtr<IPTSControlEx> spPTSEx = m_pPTS;
if (spPTSEx.p != NULL)
{
        // Call IPTSControlEx APIs
}
```

5.5.5.1 EnableMaximumLogging()

Declaration: HRESULT EnableMaximumLogging (BOOL bEnable);

Parameters: bEnable: A Boolean that specifies whether maximum logging has to be enabled or disabled.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function enables/disables the maximum logging. When maximum logging is enabled, PTS sends all logging messages to the object exposing IPTSControlClientLogger interface. If the maximum logging is disabled, only the logging messages enabled in PTS.ini file will be sent to the object exposing IPTSControlClientLogger interface.

5.5.5.2 SetPTSCallTimeout()

Declaration: HRESULT SetPTSCallTimeout (DWORD dwTimeout);

Parameters: dwTimeout: Specifies the PTS call timeout in milliseconds.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function enables automation clients to set a timeout period for the RunTestCase() calls to PTS. If, during this period, no communication is received from PTS in the form of ImplicitSend or Log messages, and the call to PTS does not complete, the PTSCONTROL_E_TESTCASE_TIMEOUT error will be returned by the RunTestCase() API call.

5.5.5.3 SaveTestHistoryLog()

Declaration: HRESULT SaveTestHistoryLog (BOOL bSave);

Parameters: bSave: Specifies test case history logs should be created/updated by the test execution.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function enables automation clients to specify whether test logs have to be saved in the corresponding workspace folder. Please keep in mind that automated PTS execution may generate significant amounts of logs and slow down the OpenWorkspace() API call execution.

5.5.6 IPTSControlDeviceSelector Interface APIs

All APIs described in this section are exposed by IPTSControlDeviceSelector interface. A pointer or reference to that interface must be obtained from the automation object to use these APIs:

```
CComQIPtr<IPTSControlDeviceSelector > spDeviceSelector = m_pPTS;
if (spDeviceSelector.p != NULL)
{
    // Call IPTSControlDeviceSelector APIs
}
```

5.5.6.1 GetDeviceList()

Declaration: HRESULT GetDeviceList(VARIANT *pvDevices);

Parameters: pvDevices: A pointer to a VARIANT object that will receive the list of all available PTS radio devices. The device list will be packaged into a safe array type object.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function facilitates discovery of all available radio devices that can be used as a PTS endpoint device. The strings describing devices may have one of the following formats:

- USB:InUse:<Unique ID> - BR/EDR/LE PTS dongle that is free and available for connection
- USB:Free:<Unique ID> - BR/EDR/LE PTS dongle that is in use by another instance of PTS and not available for connection
- COM:<Unique ID> - LE-only PTS dongle that is available for connection

5.5.6.2 SelectDevice()

Declaration: HRESULT SelectDevice(LPCWSTR pszDevice);

Parameters: pszDevice: A Unicode character string that contains the name of a device that was discovered using GetDeviceList().

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function connects PTS to the specified radio device. The connection must be established before executing any test cases.

Please note that when attempting to connect to a BR/EDR/LE dongle only available devices (identified by the substring “:Free” in the device’s descriptive name) should be used.

Passing an empty string as the device name will disconnect PTS from the currently connected endpoint device.

Connecting to a radio device using the PTS User Interface executes a device firmware version check and offers a firmware upgrade if a newer version is available. This check is disabled during automated PTS operation. Please periodically verify that the radio device firmware version is up to date using the PTS User Interface

5.5.6.3 GetSelectedDevice()

Declaration: HRESULT GetSelectedDevice(LPWSTR *ppszSelectedDevice);

Parameters: ppszSelectedDevice: A pointer to a Unicode string pointer that will receive the name of the current PTS endpoint device.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function retrieves the name of the current PTS endpoint device in the format described in Section 5.5.5.4. If PTS is not connected to a radio, empty string will be returned.

5.6 General information functions

5.6.1 GetPTSBluetoothAddress()

Declaration: HRESULT GetPTSBluetoothAddress(ULONGLONG* pullBdAddr);

Parameters: pullBdAddr: A pointer to a 64 bit unsigned integer that receives the BDADDR of the PTS endpoint device.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

This function retrieves the Bluetooth Device Address (BDADDR) of the endpoint device that is being used by PTS.

A 64 bit value is returned even though a BDADDR is only 48 bits in length. The BDADDR is located in the least significant 48 bits (six bytes) and the upper two bytes will have a value of 0x0000.

It should be noted that the Bluetooth Device Address may not be immediately available after PTS is started. During PTS startup, a number of different things happen including communicating with the endpoint device to determine its BDADDR. GetPTSBluetoothAddress() will return a status of PTSCONTROL_E_BLUETOOTH_ADDRESS_NOT_FOUND if it is called too early.

The clientShowBdAddress() function in the PTSControlClient sample program demonstrates a way to handle this situation by waiting up to 15 seconds for the BDADDR to become available.

5.6.2 GetPTSVersion()

Declaration: HRESULT GetPTSVersion(DWORD* pPTSVersion);

Parameters: pPTSVersion: A pointer to a 32 bit unsigned integer that receives the version number of PTS.exe.

Return values: A value greater than or equal to zero if successful.
A value less than zero if not successful. For a list of error codes specific to the PTS Control API see section 7 (“API Error Codes”).

Returns the version of PTS as a four byte value packed into a 32 bit unsigned integer. Each byte represents one piece of a standard Windows version number:

- Byte 3: Major version number
- Byte 2: Minor version number
- Byte 1: Update release number
- Byte 0: Build sequence number

For example, for PTS version 4.5, update 2, build number 6 (4.5.2.6), the value returned from GetPTSVersion() would be 0x04050206.

6 Sample program – PTSControlClient

PTSControlClient is provided as a part of PTS to serve as an example of using the PTS Control API. It is provided in both source code form and as a ready to run executable.

The source code is found in

C:\Program Files [(x86)]\Bluetooth SIG\Bluetooth PTS\SampleCode\PTSControlClient

and the executable can be found at

C:\Program Files [(x86)]\Bluetooth SIG\Bluetooth PTS\bin\PTSControlClient.exe

6.1 Preparing to use PTSControlClient

There are two steps that need to be performed before using the PTSControlClient.

6.1.1 Creating a Workspace

The PTSControlClient does not demonstrate the CreateWorkspace() function. Instead, it requires a pre-existing Workspace.

A new Workspace can be created using the PTS User Interface. If a suitable Workspace already exists, it can also be used.

Once the Workspace to be used is configured as needed, exit PTS so that the PTSControlClient can launch it in COM Server mode.

6.1.2 Create a Test Script

Note: The Test Script referred to here is specific to the PTSControlClient and should not be confused with the PTS Test Scripting feature described in the "Scripting" reference document.

Test Scripts for the PTSControlClient are disk files formatted in XML. Any convenient method can be used to create a Test Script – Notepad, an XML editor such as XML Notepad, Excel or any other mechanism that can be used to create a text file in the XML format.

An example of a plain text PTSControlClient Test Script can be found at

C:\Program Files [(x86)]\Bluetooth SIG\Bluetooth PTS\Documentation\Automation\TestScriptSample.xml

For applications such as Excel, the Test Script can be edited and exported as XML data. As a starting point, open

C:\Program Files [(x86)]\Bluetooth SIG\
Bluetooth PTS\Documentation\Automation\TestScriptTemplate.xlsx

and follow the instructions.

6.1.3 Test Script format

- The contents of the script are enclosed in an “<AUTOMATION>”, “</AUTOMATION>” tag pair.
- The Test Script must be given a name that is enclosed in a “<Name>”, “</Name>” tag pair.
- The file path to the Workspace to be used is enclosed in a “<Workspace>”, “</Workspace>” tag pair. A full file path should be used for the Workspace.

  Only one Workspace may be specified. If more than is present, only the first one is used.
- The name of a Project from the workspace is enclosed in a “<Testsuite>”, “</TestSuite>” tag pair.

  Only one Project may be specified. If more than is present, only the first one is used.
- Other tag pairs that may exist in the file are ignored. Specifically, the sample Test Scripts mentioned above include a “<Version>”, “</Version>” tag pair but it is not currently used.
- Any number of Test Cases from the selected Project are enclosed in a “<Testcase>”, “</Testcase>” tag pair, one for each Test Case. The Test Cases will be executed in order in which they appear in the file.

For example, script.xml may contain

```
<AUTOMATION>
        <Name>SampleTest</Name>
        <Workspace>C:\Program Files\Bluetooth SIG\My Workspaces\Sample\Sample.pqw</Workspace>
        <Testsuite>IOPT</Testsuite>
                <Testcase>TC_COD_BV_01_I</Testcase>
                <Testcase>TC_COD_BV_01_2</Testcase>
</AUTOMATION>
```

6.2 Running the Test Script

Once the Test Script is ready, it can be executed by running PTSControlClient, giving the name of the Test Script as a command line parameter. The path to the Test Script does not need to be a full path since the file is only processed by the PTSControlClient application.

“C:\Program Files [(x86)]\Bluetooth SIG\Bluetooth PTS\bin\PTSControlClient.exe” script.xml

7 API Error Codes

7.1 PTSCONTROL_E_GUI_UPDATE_FAILED (0x849C0001)

ICS and/or IXIT changes that result from calling UpdatePics() or UpdateIxitParam() need to be communicated to the appropriate Executable Test Suite DLLs. This error occurs when the update process fails.

7.2 PTSCONTROL_E_PTS_FILE_FAILED_TO_INITIALIZE (0x849C0002)

This error will be returned by CreateWorkspace() if the ICS file specified in the pszPathOfPtsFile is invalid or cannot be found.

7.3 PTSCONTROL_E_FAILED_TO_CREATE_WORKSPACE (0x849C0003)

CreateWorkspace() will return this error code if there is a problem creating the Workspace.

7.4 PTSCONTROL_E_CLIENT_LOG_NOT_EXPECTED_TO_FAIL (0x849C0004)

This error can be returned from RunTestCase() when a call to the “logger” function (IPTSControlClientLogger::Log()) returns a failure status.

7.5 PTSCONTROL_E_FAILED_TO_OPEN_WORKSPACE (0x849C0005)

OpenWorkspace() will return this status code when it is unable to open the Workspace specified in the pszPathOfWorkspace parameter.

7.6 PTSCONTROL_E_PROJECT_NOT_FOUND (0x849C0010)

PTSCONTROL_E_PROJECT_NOT_FOUND is returned when the Project index value to GetProjectName() is out of range, or by GetProjectVersion(), the Test Case functions, and the ICS/IXIT update functions when the named Project is not in the Workspace.

7.7 PTSCONTROL_E_TESTCASE_NOT_FOUND (0x849C0011)

This value is returned when the Test Case index value to GetTestCaseName() or GetTestCaseDescription() is out of range, or, when the Test Case name supplied to IsActiveTestCase() does not exist.

7.8 PTSCONTROL_E_TESTCASE_NOT_STARTED (0x849C0012)

Returned by RunTestCase() when it encounters a problem trying to start the execution of the specified Test Case.

7.9 PTSCONTROL_E_INVALID_TEST_SUITE (0x849C0013)

This value is returned from RunTestCase() or IsActiveTestCase() if the data for the selected Project is invalid in some way.

7.10 PTSCONTROL_E_PTS_VERSION_NOT_FOUND (0x849C0014)

Returned by GetPTSVersion() when it is unable to determine the version of PTS.exe.

7.11 PTSCONTROL_E_PROJECT_VERSION_NOT_FOUND (0x849C0015)

Returned by GetProjectVersion() when the selected project is not present in the current Workspace.

7.12 PTSCONTROL_E_TESTCASE_NOT_ACTIVE (0x849C0016)

This value is returned from RunTestCase() when the selected Test Case is not active (disabled) in the selected Project.

7.13 PTSCONTROL_E_TESTCASE_TIMEOUT (0x849C0017)

This value is returned from RunTestCase() when the API call timeout is greater than zero and expires. See Section 5.5.5.2 for additional information.

7.14 PTSCONTROL_E_INVALID_IXIT_PARAM_VALUE (0x849C0020)

This value is returned from UpdateIxitParam() when the character string containing the new parameter value contains characters that are not valid for the IXIT item data type.

This value is also returned for OCTETSTRING values that contain only hexadecimal characters, but are not of an even length. (OCTETSTRINGs require two characters for each byte of data.)

7.15 PTSCONTROL_E_IXIT_PARAM_NOT_CHANGED (0x849C0021)

UpdateIxitParam() can return this error status when there is a problem updating a IXIT value.

7.16 PTSCONTROL_E_IXIT_PARAM_UPDATE_FAILED (0x849C0022)

UpdateIxitParam() can return this error status when there is a problem updating a IXIT value.

7.17 PTSCONTROL_E_IXIT_PARAM_NOT_FOUND (0x849C0023)

UpdateIxitParam() returns this value when the specified IXIT item is not defined for the selected Project.

7.18 PTSCONTROL_E_TEST_SUITE_PARAM_UPDATE_FAILED (0x849C0024)

UpdatePics() or UpdateIxitParam() can return this error status when there is a problem updating a ICS or IXIT value.

7.19 PTSCONTROL_E_PICS_ENTRY_UPDATE_FAILED (0x849C0030)

UpdatePics() can return this error status when there is a problem updating a ICS value.

7.20 PTSCONTROL_E_PICS_ENTRY_NOT_FOUND (0x849C0031)

UpdatePics() returns this value when the specified ICS item is not defined for the selected Project.

7.21 PTSCONTROL_E_PICS_ENTRY_NOT_CHANGED (0x849C0032)

UpdatePics() can return this error status when there is a problem updating a ICS value.

7.22 PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_NOT_REGISTERED (0x849C0040)

This value is returned from UnregisterImplicitSendCallback() when no callback is currently registered.

7.23 PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_ALREADY_REGISTERED (0x849C0041)

This value is returned from RegisterImplicitSendCallback() when a callback is currently registered.

7.24 PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_NOT_EXPECTED_TO_FAIL (0x849C0042)

This error can be returned from RunTestCase() when a call to the Implicit Send handler (IPTSImplicitSendCallback::OnSend()) returns a failure status.

7.25 PTSCONTROL_E_BLUETOOTH_ADDRESS_NOT_FOUND (0x849C0043)

GetPTSBluetoothAddress() will return this error if it is unable to determine the Bluetooth Device Address of the PTS endpoint device. This usually means that the endpoint device is not connected to the computer or has the wrong device driver attached to it.

On some occasions, this error may indicate that the call to GetPTSBluetoothAddress() occurred while PTS was still initializing. In this case, wait about 10 to 15 seconds after starting PTS manually or via CoCreateInstance() before calling GetPTSBluetoothAddress().

7.26 PTSCONTROL_E_INTERNAL_ERROR (0x849C0044)

This status value represents a general unspecified failure during a call to one of the PTS Control API functions.

Currently, only GetPTSBluetoothAddress() may returned this value, but other functions could use it in the future.

7.27 PTSCONTROL_E_DEVICE_NOT_AVALABLE (0x849C0045)

This error can be returned from SelectDevice() when attempt is made to connect to a device that is already connected to another instance of PTS or does not exist.

7.28 PTSCONTROL_E_FUNCTION_NOT_IMPLEMENTED (0x849C0099)

This status value is used by functions that are defined in the PTS Control API, and are available to be called, but have not actually been implemented.

Currently, StopTestCase() is the only function the can be called but has no implementation.

7.29 Other error codes

7.29.1 E_NOINTERFACE (0x80004002)

This error is likely to be returned from CoCreateInstance() and probably indicates that the PTS Control API has not been registered with the Component Object Module manager. To correct this error

1. Open a command prompt;

2. Set the current working directory to the program directory for PTS. In a normal installation this is

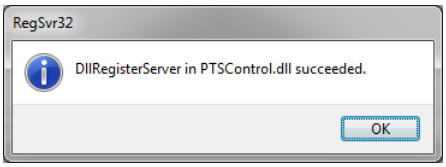
C:\Program Files [(x86)]\Bluetooth SIG\Bluetooth PTS\bin

3. Enter the following command

   Regsvr32 PTSControl.dll

4. If a message box like the one at the right appears then the PTS Control API has been successfully registered with COM.

   If a different message appears, please contact PTS Technical support for additional assistance.



7.29.2 CO_E_SERVER_EXEC_FAILURE (0x80080005)

CoCreateInstance() is likely to return this error when PTS is currently running but was not started in COM server mode. Exit the active instance of PTS and try the function again.