

Bluetooth SIG - PTS User Manual

FCC Statement:

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

- (1) This device may not cause harmful interference, and
- (2) this device must accept any interference received, including interference that may cause undesired operation.

(3) FCC ID: **FCC ID 2AECO-BTSIG15A**

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation.

This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- ___ Reorient or relocate the receiving antenna.
- ___ Increase the separation between the equipment and receiver.
- ___ Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- ___ Consult the dealer or an experienced radio/TV technician for help.

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

.

Table of Contents

1. Workspaces and Projects	1
Creating a new workspace	1
Opening an existing workspace	5
Adding a project to a workspace.	7
Deleting a workspace	8
Removing a project from a workspace	8
PTS Terminology	9
Using the Test Suite Selector	9
Using the Test Suite Selector	9
Primary windows	10
Editing the list of test suites for inclusion in the workspace	12
Filtering the "Test Suite(s)" window	16
Projects	25
Projects	25
Editing the project ICS	28
Editing the project IXIT settings	33
Test Cases	38
Displaying the purpose of a test case	38
Executing a single test case	39
Aborting test case execution	42
Link Keys and PTS Endpoint Information	42
Endpoint information	42
Deleting the current Link Key	43
PTS Program Settings	44
PTS Program Settings	44
Application settings	44
Project Settings	47
2. Automating	51
Automating PTS	51
"Operator-less Operation"	51
Automation test platforms	53
PTS test case operation	53
Implicit Send DLLs	54
Basic Information	54
Implicit Send functions	55
Message tags	59
MMI styles	61
Software build requirements	64
Activating a Custom Implicit Send DLL	64
Activating a custom Implicit Send DLL	64
Usage Notes	66
Technical Tidbits	66
Automatic dismissal of Implicit Send requests	66
ImplicitSend() function	67
TSPX_use_implicit_send	67
Sample Source Code	67
One DLL or many DLLs?	67
Hybrid environments	67
3. Extended Automating	69
Automating PTS	69
"Fully Automated Operation"	69

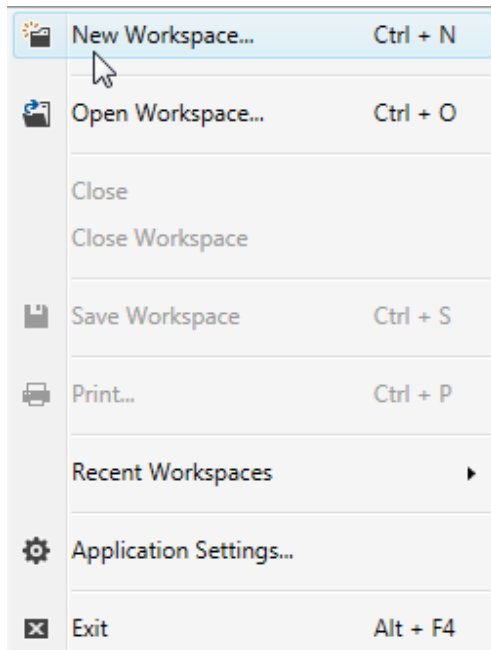
PTS and the PTS Control API	70
General Usage	70
Functions in the PTS Control API	71
Opening/Creating a Workspace	71
Working with Projects	72
Working with Test Cases	74
Working with ICS and IXIT data	78
Logging and unattended operation	80
General information functions	89
Sample Program - PTSTestClient	90
Sample Program - PTSTestClient	90
Preparing to use PTSTestClient	90
Running the Test Script	91
API Error Codes	91
Other error codes	93
4. Report Generator	95
Introduction	95
Qualification test evidence	95
Development checkpoints	95
Contents of a report	95
Creating a PTS Report	96
1. Select a workspace	96
2. Start the report generator	97
3. First time use of the report generator	97
4. Select the device description	99
5. Selecting the test case results to be used in the report	100
8. Including test execution logs in the report	103
9. Generating the report	104
Adding and Deleting Device Descriptions	105
Adding a device description	105
Deleting a device description	106
Reviewing and Editing the Test History	108
Reviewing and Editing the Test History	108
Viewing a test case execution log	109
Selecting individual test case results to be deleted	110
Selecting older test results for deletion	110
Deleting test case results	111
5. Scripting	113
Automating PTS	113
"Scripted Operation"	113
Creating an initial Test Script	113
Adding Test Cases to the Test Script	115
Executing a Test Script	116
Stopping Test Script execution	117
Editing a Test Script	117
Editing a Test Script	117
Removing a Test Case from the Test Script	117
Changing the order of execution	120
Adding a Test Case to the Test Script	122
6. Logging	123
Introduction	123
Output Window	123
Test Case History Tool Window	124

Test Execution Log	125
Test Execution Log	125
Format of the execution log	126
Interesting events	127
Selecting the events to be logged	130
"Run-time" vs. deferred logging	132
PTS Protocol Viewer	133
PTS Protocol Viewer	133
The Protocol Viewer "stack"	133
Starting the PTS Protocol Viewer	135
Saving and viewing protocol traces	135
Verdict Determination	135
Verdict Determination	135
 7. Verdict Determination	 137
 8. Index	 139

Workspaces and Projects

Creating a new workspace

The first step in creating a new workspace is to select the “New Workspace” item on the PTS “File” menu. A wizard consisting of three dialogs will start when “New Workspace” is selected.



Selecting the Implementation Under Test

The first dialog is used to select the device to be tested. There are three ways to specify the device:

- If the device appears in the list at the left hand side of the dialog, it may be selected by clicking on its entry.
- If the device is currently discoverable, a search for it may be started by clicking the “Search” button. When the device appears in the list, it may be selected by clicking on its entry.
- The Bluetooth Device Address (BD_ADDR) may be entered directly in the box labeled “IUT Device Address”.

After the device has been selected using one of the above methods, click the “Next >” button to proceed to the next step.

- The device to be tested can be specified at a later time if this is needed. To do this, enter a dummy BD_ADDR such as “000000000000” and click “Next >”. The BD_ADDR for the device is a IXIT item and can be edited before testing is started.
- The list at the left hand side of the dialog may be cleared by click the “Clear Search Results” button.
- When the “Search” button is clicked, the label on the button changes to “Stop Searching”. The search operation will continue until either the “Next >” or “Stop Searching” button is clicked.

- The devices found during the search operation may be filtered by selecting an appropriate Major Device Class and/or Service Class using the radio buttons in the lower right hand corner. This will not remove any devices that are already in the list, it will simply discard devices found during the search operation that do not match the selected criteria.

The selection of devices to be filtered is based on the Class Of Device value that each device reports in its response to the Inquiry Request sent by the PTS.

“Import ICS from TPG”

An important part of preparing to test a given device is to select the profiles and protocols it supports, and to edit the ICS information for each of those items. This process can be simplified if a declaration of the device's features and functions already exists.

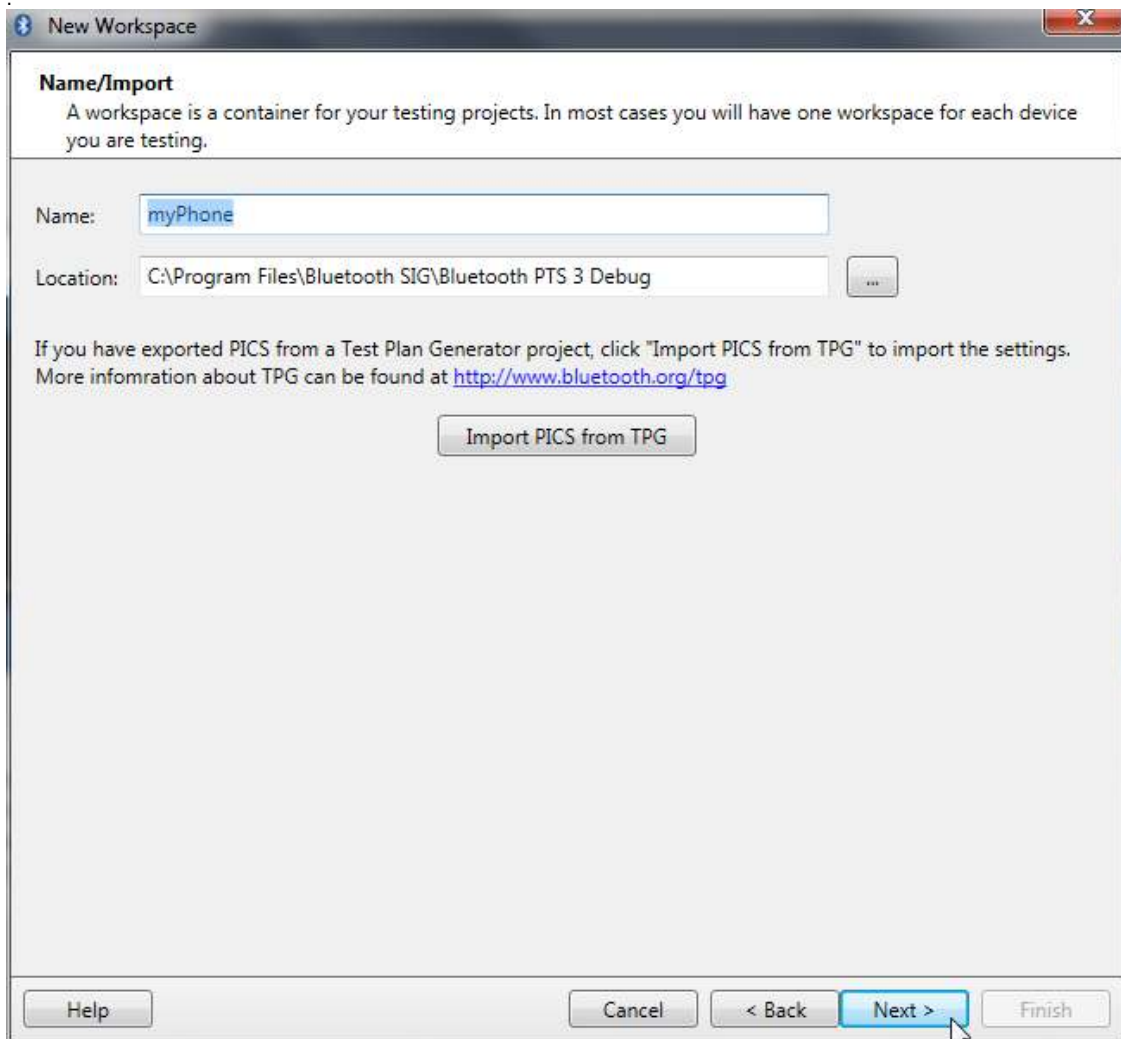
One of the steps in getting a device ready for qualification is to declare the profiles and protocols it supports using the online Test Plan Generator (TPG). One of the steps in completing this declaration is to edit the ICS information for the device.

The ICS information stored in the TPG may be exported to a file on your computer. That file may then be imported to PTS clicking the “Import ICS from TPG” button.

The ICS information for previously qualified devices that were qualified using PRD 2.0 or later may also be exported from the Design/Product Listings area of Bluetooth.org for later import into PTS.

Clicking the “Import ICS from TPG” button will open the “Import Test Plan” dialog. This is a normal file selection dialog which may be used to locate and select the file containing the ICS information exported from the TPG or a Qualified Device Listing.

The “Import ICS from TPG” function may also be used at later time to update the ICS used by PTS when the declaration for the device is updated in the TPG. See [Adding a project to a workspace](#) for more information



Naming the workspace

Each workspace is given a name. The name can be anything that is meaningful to you, such as the product name of the device, the internal codename for the device, the software version found in the device or some combination of the above. There are two restrictions on the name of the workspace. First, the name needs to be different than the names of other workspaces that you have created. Additionally, the characters used in the name of the workspace are limited to those that may be used in a disk file or folder name. (The collection of data representing the workspace is stored in a disk file folder whose name is the name of the workspace.)

The name of the workspace is entered in the box labeled "Name". By default, the name of the workspace will be the "Device Name" shown in the previous dialog.

Please note that importing ICS information as described above will change the workspace name to the name found in the imported data. It's a good idea to execute "Import ICS from TPG" before entering the name of the workspace.

Normally, the folder and data files for a workspace are created in a subfolder of your PTS installation. The usual location for workspaces is:

C:\Program Files\Bluetooth SIG\Bluetooth PTS\My Workspaces

An alternate location may be specified in the box labeled "Location". The "..." button will initiate a folder browse operation which may be used to select an alternate location for the workspace.

After the profile and protocol declaration for the device has been selected, if desired, and the workspace has been given a name (and possible alternate storage location), click the "Next >" button.

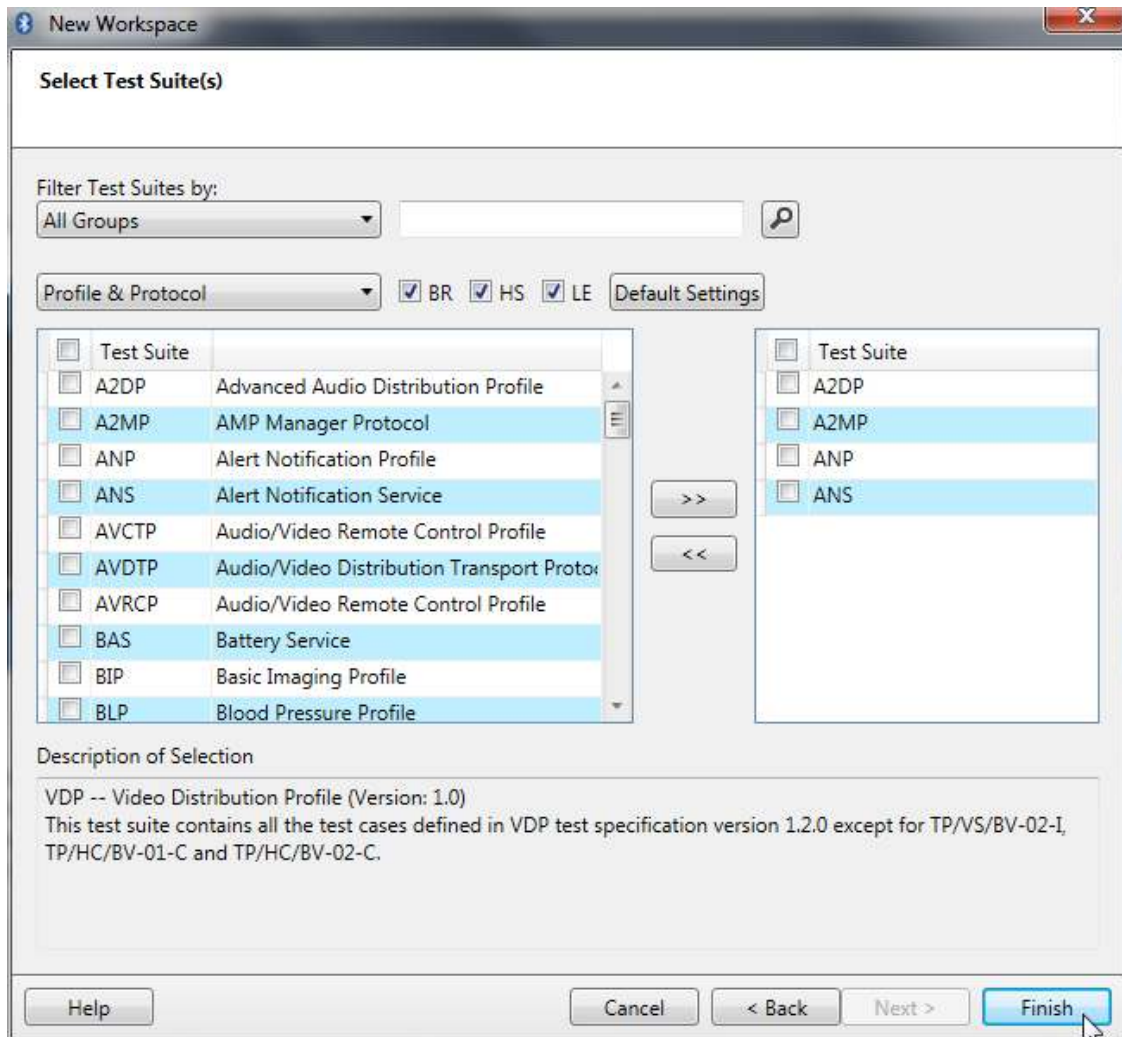
Selecting profiles and protocols

The final step in creating a new workspace is to select the profiles and protocols to be tested. If the "Import ICS from TPG" function was used in the previous step, the profiles and protocols supported by the device will already be selected.

The list of selected profiles and protocols may be changed before the workspace is created. See [Using the Test Suite Selector](#) for more information.

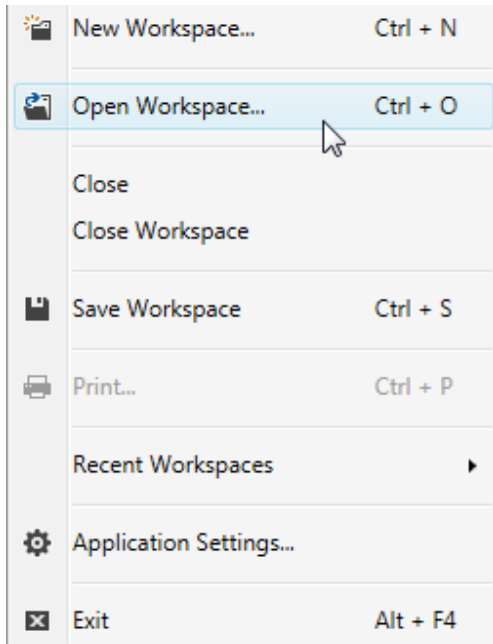
Additional profile or protocol projects may be added to the workspace at a later time. [Adding a project to a workspace](#) describes this process.

After you have selected the profiles and protocols that you wish to test, click "Finish" to create the workspace. The creation of the workspace may take a little time, so be patient.

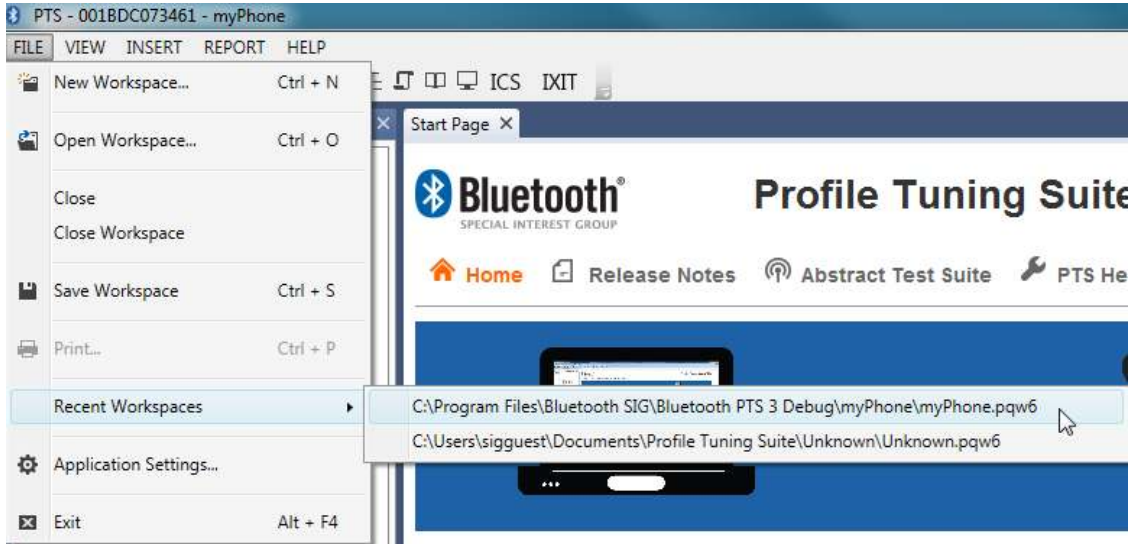


Opening an existing workspace

An existing workspace may be opened in one of two ways. The first method is to select “Open Workspace...” from the “File” menu.

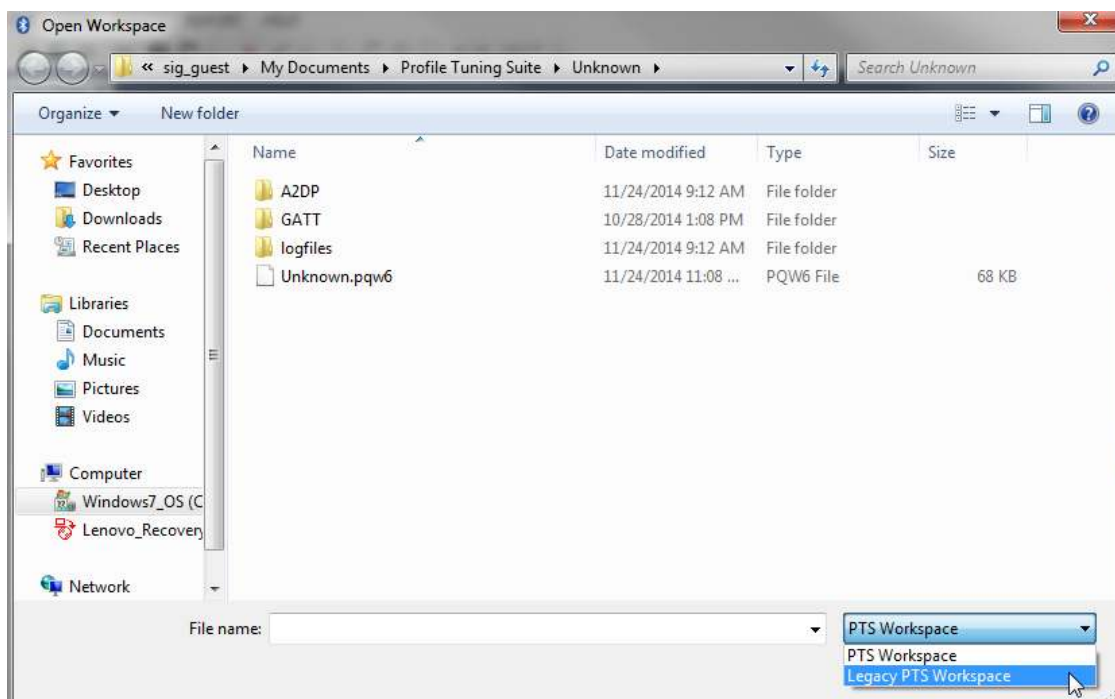


PTS also keeps a list of recently used workspaces. This list may be accessed by selecting "Recent Workspaces" on the "File" menu.



Please note that opening a workspace may take a little time, so be patient.

In addition to opening a PTS Workspace, you can also open a Legacy PTS Workspace by selecting Legacy PTS Workspace in the File name drop down menu.



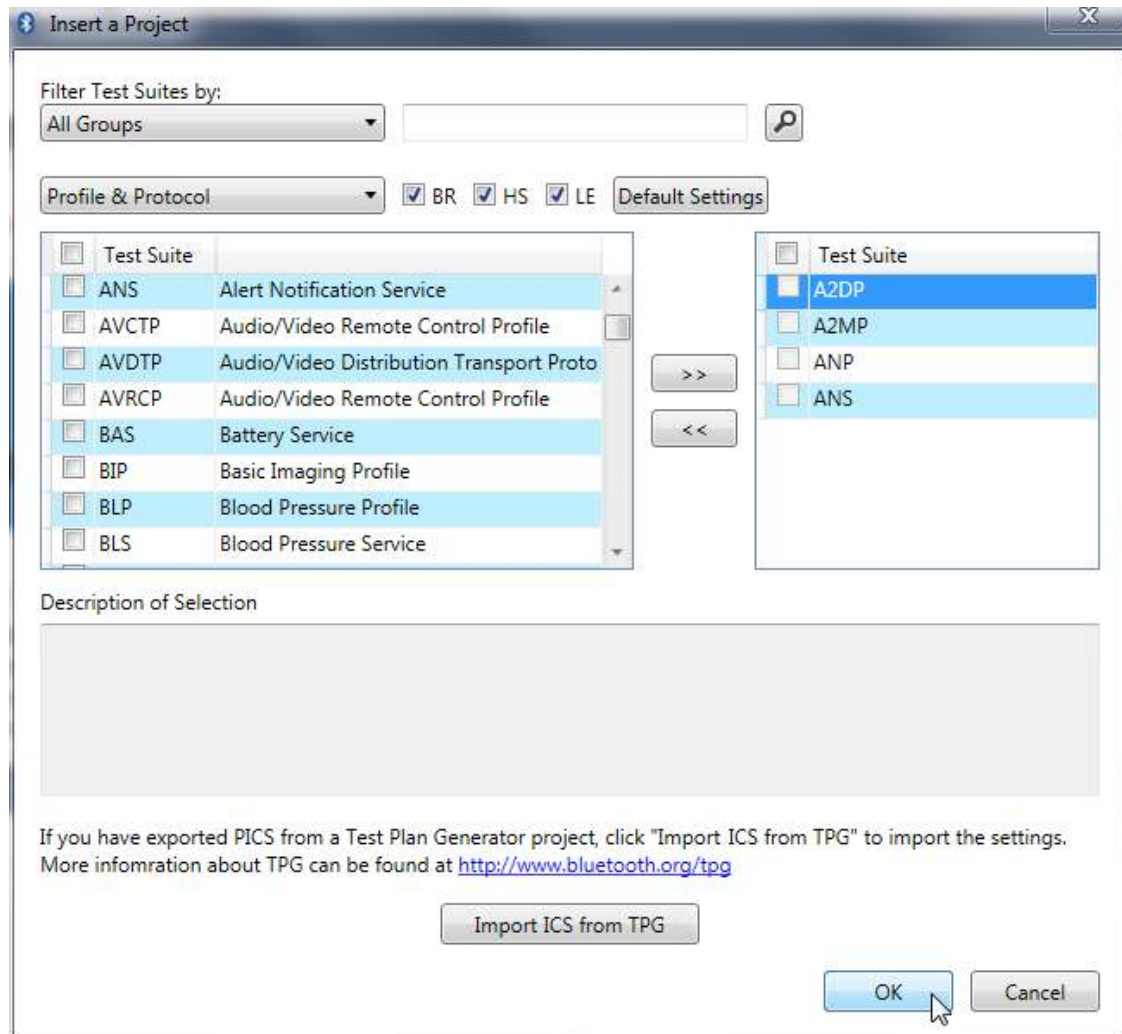
Adding a project to a workspace.

Additional projects – profiles or protocols – may be added to a workspace at any time by selecting the “Project” item on the “Insert” menu. The “Test Suite Selector” dialog will appear allowing additional projects to be selected. (See [Using the Test Suite Selector](#)).



Profiles that are already present in the workspace will appear “greyed out” in the “Your Suite(s)” window and cannot be removed.

The “Import ICS from TPG” function may also be used at this time. If you have updated your device declaration in the TPG by changing the ICS settings or adding additional profiles or protocols, the updated declaration may be imported to synchronize your settings in PTS to match the settings you are using in the TPG.



Deleting a workspace

A workspace may be deleted using the following procedure:

1. Exit the PTS application.
2. Using Windows Explorer, locate the folder that contains the various data files used to represent the project.
3. Right click on the folder and select "Delete" from the popup menu.

Removing a project from a workspace

At the present time there is no supported mechanism for deleting a project from a workspace.

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 (Profile 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 (Profile 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.)

Using the Test Suite Selector

Using the Test Suite Selector

There are currently over three dozen Bluetooth profiles and protocols available to be tested by PTS. The task of determining which of the available test suites that should be used can be a little daunting. To make things easier, the Test Suite Selector has been created.

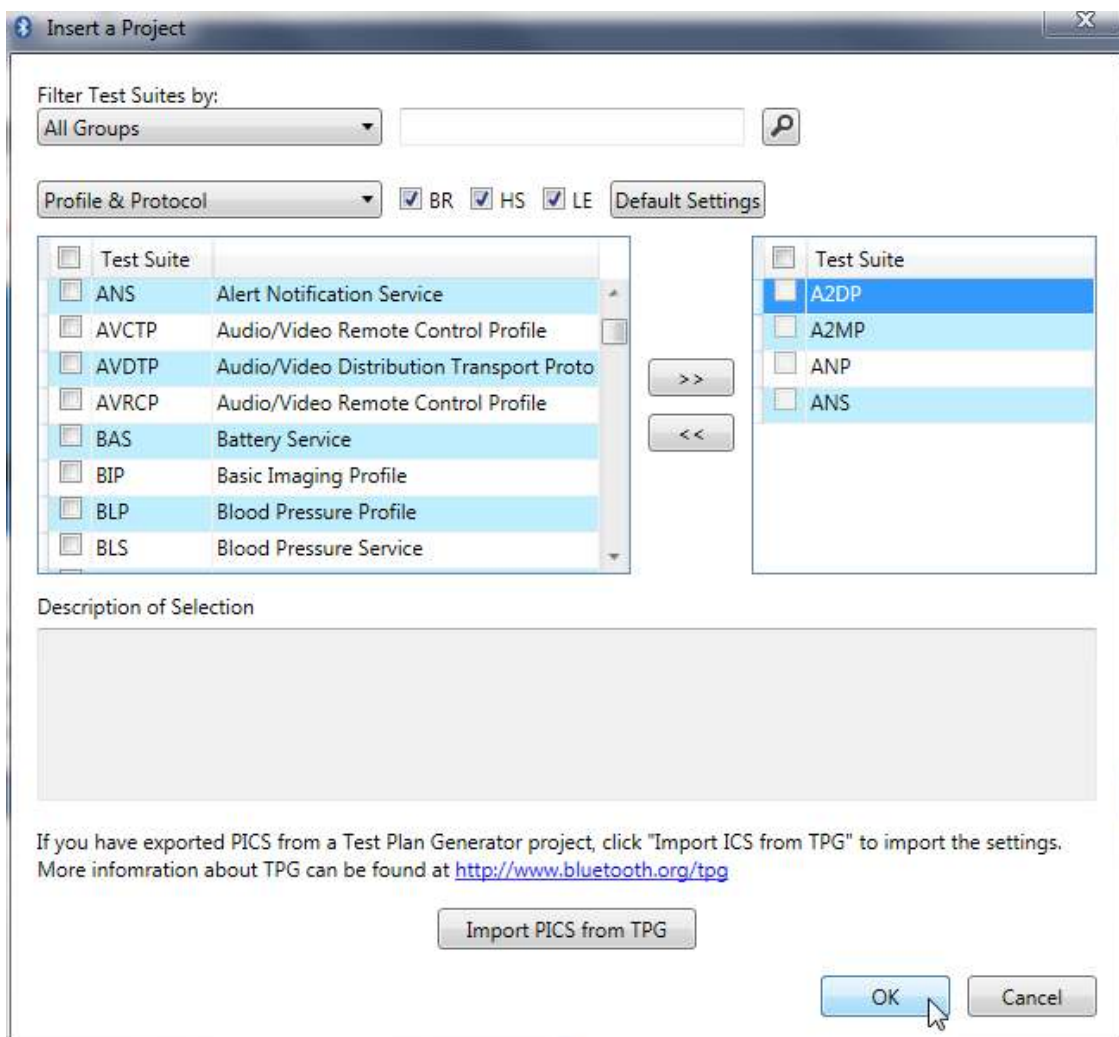
The Test Suite Selector dialog displays all of the profile and protocol test suites available in PTS. The list may be filtered in a number of ways to simplify the process of locating the test suites of interest.

The Test Suite Selector will appear in two different scenarios:

1. As the last step when [creating a new workspace](#);
2. When [adding a project to an existing workspace](#).

Primary windows

There are two primary windows in the Test Suite Selector dialog: “Test Suite(s)” on the left and “Your Suite(s)” on the right.



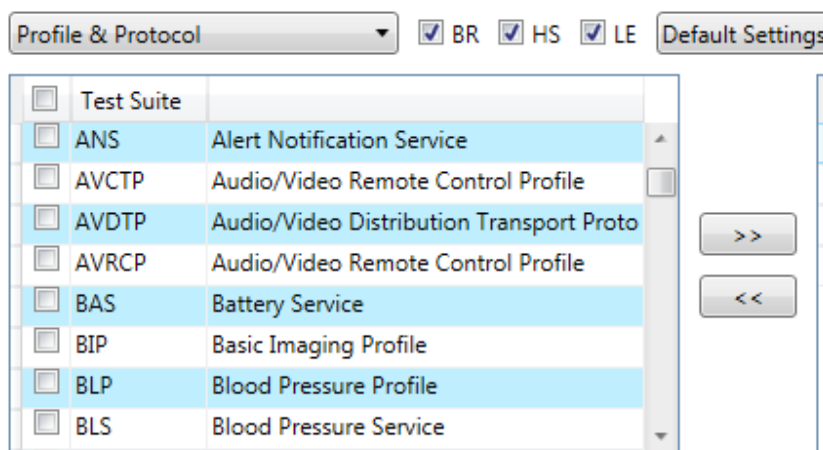
“Test Suite(s)” window

The “Test Suite(s)” window contains a list of the profile and protocol test suites that are available to be added to a workspace. When the “Your Suite(s)” window is empty, this list will contain all of the test suites that are available in the current installation of PTS.

Test suites that are currently present in the workspace, or that have been selected for addition to the workspace will appear in the “Your Suite(s)” window and not in the list of available test suites.

Each test suite shown in the list is identified by its full name and common acronym.

Some test suites are considered to be in a “beta test” state. These test suites are generally new, or contain test cases which only a few (or maybe no) Bluetooth devices are available. Test suites in “beta test” are indicated by “(Beta)” following the profile or protocol name.



“Your Suite(s)” window

This window contains a list of test suites that are already present in the current workspace, or are waiting to be added to the workspace. Items that are “greyed out” are already present; items in normal text have been selected for addition to the workspace. In the picture on the previous page

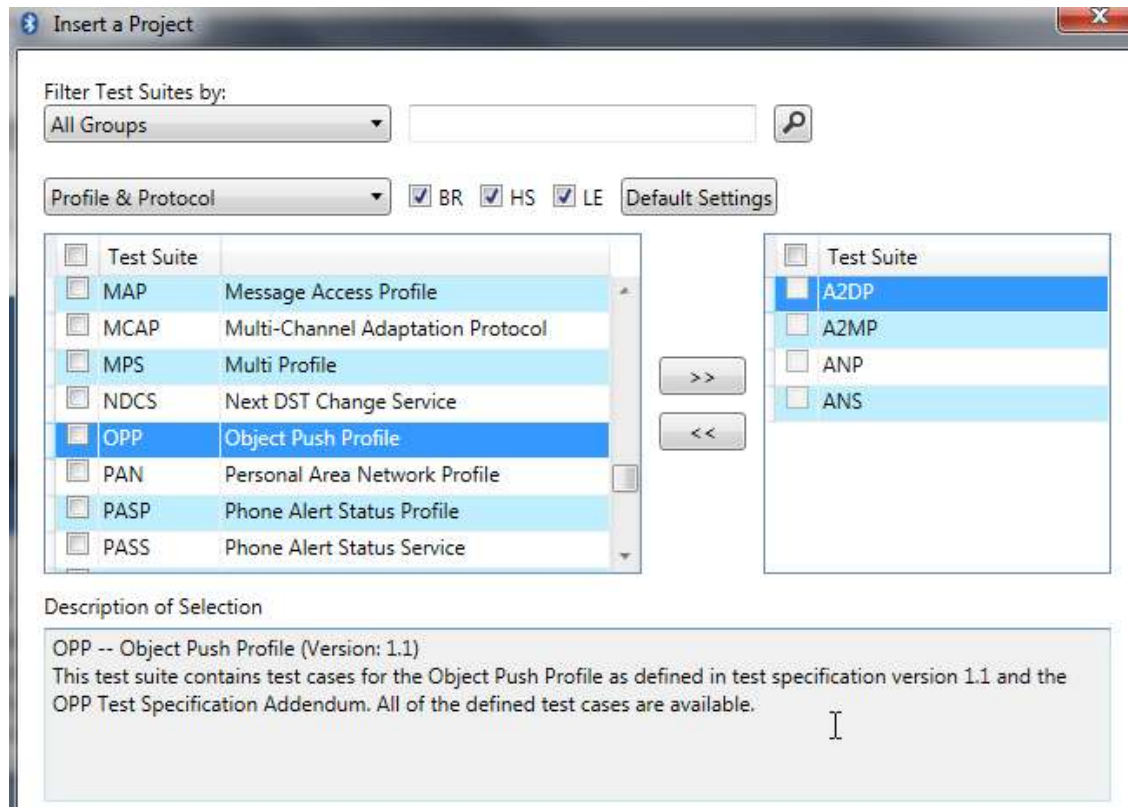
- The OPP test suite has been selected for addition;
- A2DP, AVRCP, HFP15 and PBAP are already present in the workspace.

Test suite description

The “Description of Selection” window at the bottom of the Test Suite Selector will display descriptive information about the item that is currently selected in either the “Test Suite(s)” or “Your Suite(s)” window.

The description may include the profile or protocol version supported by the test suite, the applicable test specification version, and any special notes about usage or test case coverage.

Note that no description will be shown if more than one item is selected in either window.



Editing the list of test suites for inclusion in the workspace

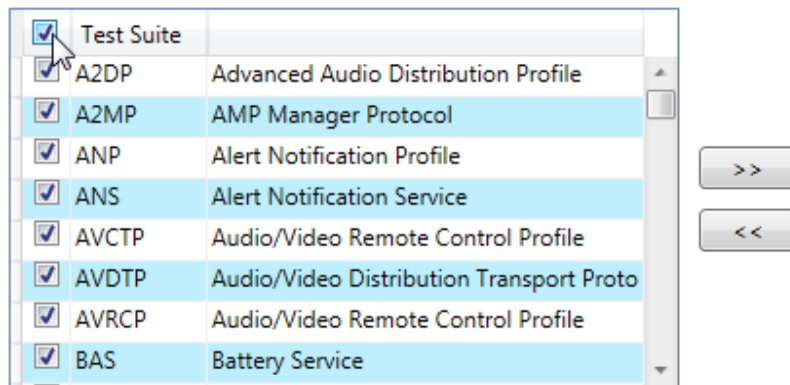
Adding test suites to “Your Suite(s)”

There are two ways to select test suites for inclusion in the workspace. The first, and possibly the easiest method, is to locate a desired test suite in the “Test Suite(s)” window and double-click on it. Doing this will immediately move the selected suite to the “Your Suite(s)” window.

Sometimes, it may be desirable to choose a set of test suites and select them all at once. To do this,

- Control-click on each of the test suite names of interest;
- Choose a contiguous block of suites by clicking on the first test suite name, followed by a shift-click on the last name.

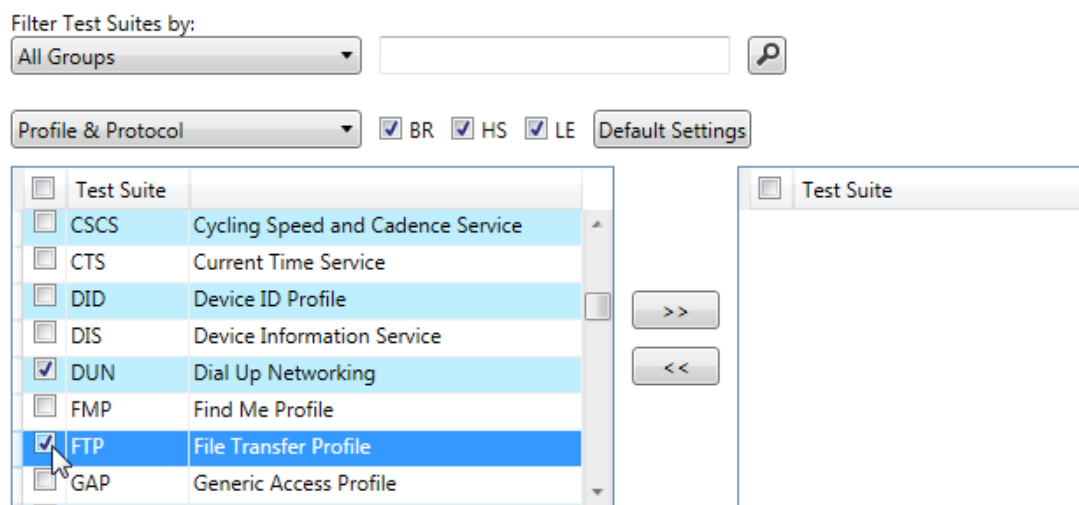
The “Select all” checkbox may be used as a convenient shortcut if all of the available test suites are to be selected.



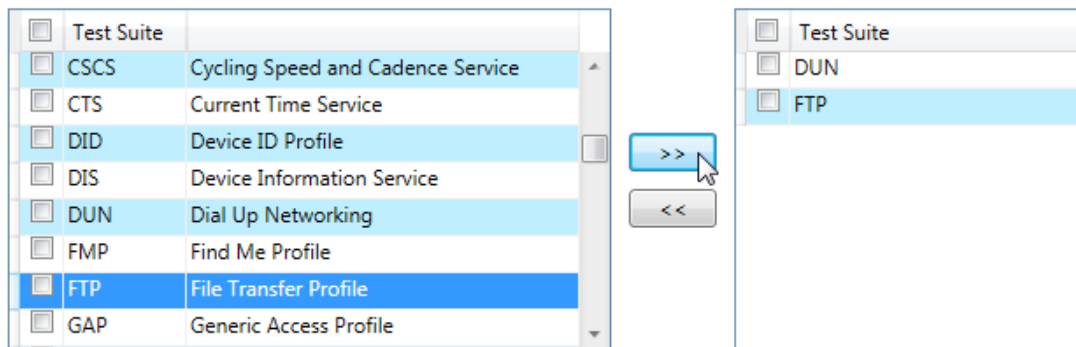
Note that control-clicking on a selected item in the "Test Suite(s)" window will un-select it.

After a set of selections has been made, the ">>" button between the two windows will be active. Press the ">" button to copy the selections to the "Your Suite(s)" window.

Before pressing ">>":



After pressing ">":



Notice that “DUN” and “FTP” moved from the “Test Suite(s)” window to the “Your Suites(s)” window after the selection was executed.

Additional selections may be made until the desired set of test suites appears in the “Your Suite(s)” window.

Test suites that are not available selection

At times some of the test suites listed in the “Test Suite(s)” window will be “greyed out” and cannot be selected.

Removing a test suite from “Your Suite(s)”

The “Your Suite(s)” window behaves identically to the “Test Suite(s)” window. Items in this window can be easily moved back to the “Test Suite(s)” window by double-clicking on them.

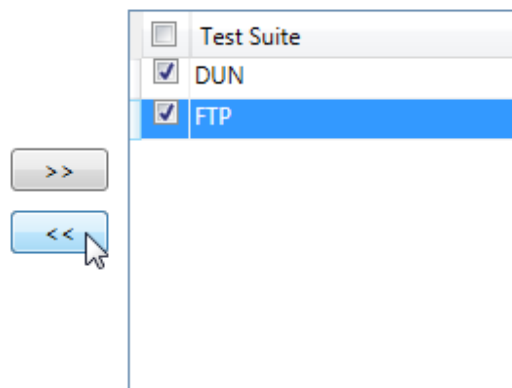
Otherwise, the set selection functions described above may be used:

- To select an item for removal, control-click on its name;
- To select a contiguous group of test suites for removal, click on the first name in the group and then click on the last name while holding down the “Shift” key (shift-click);
- To select all of the suites shown in “Your Suite(s)”, place a checkmark in the “Select All” checkbox.

After a set of selection selections have been made in the “Your Suite(s)” window, the “Remove” button will become active. Press this button to move the selected items back to the “Test Suite(s)” window.

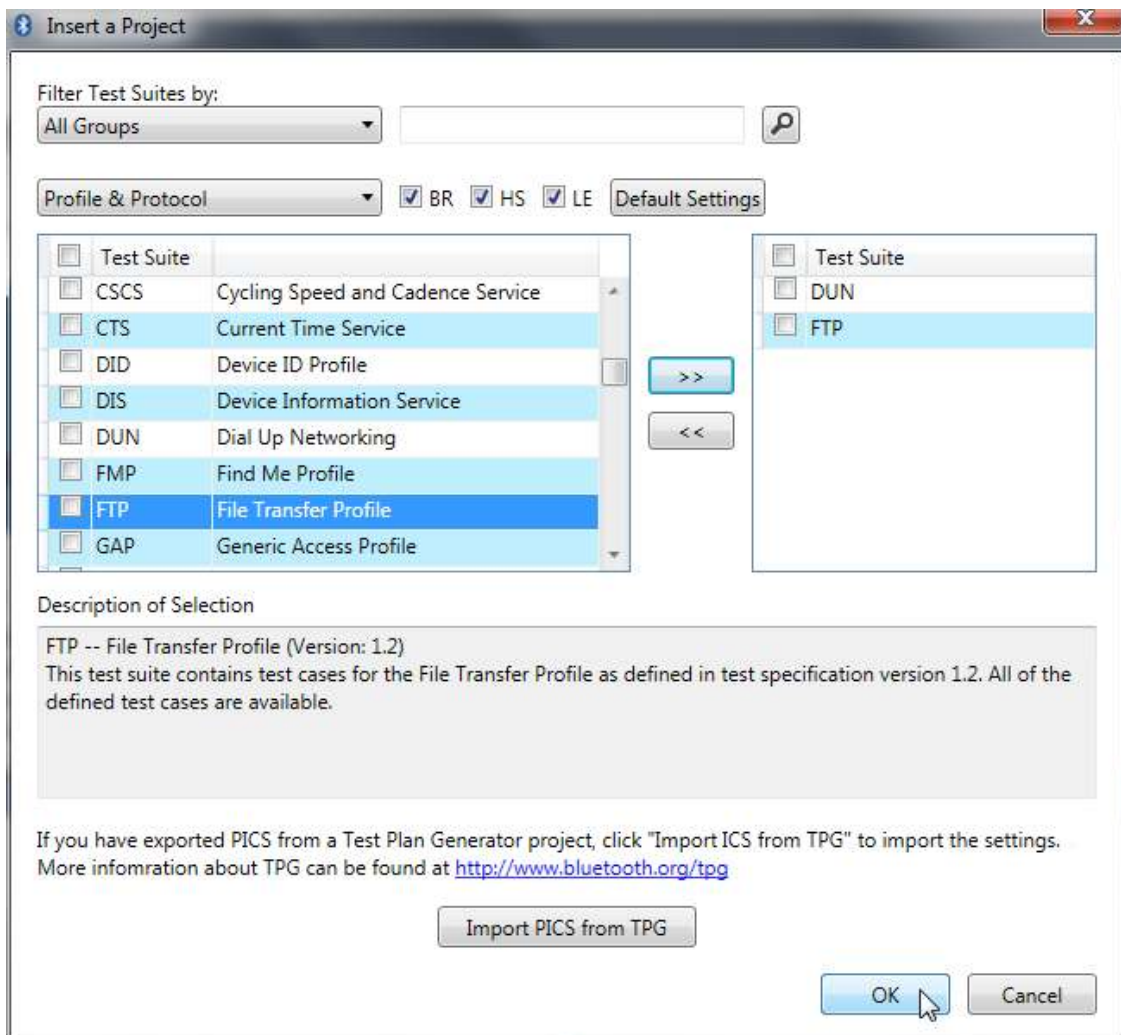
As noted above, items which are “greyed out” are currently a part of the workspace and cannot be removed.

Items that would be hidden by the filtering functions in the “Test Suite(s)” window will not appear after they have been removed. They have been moved back to “Test Suite(s)”, they just will not be seen until the filtering condition has been removed.



Completing the process

Once the desired set of profile and protocol test suites have been selected, and are appearing in the “Your Suite(s)” window, press the “Finish” button to create or update the workspace.



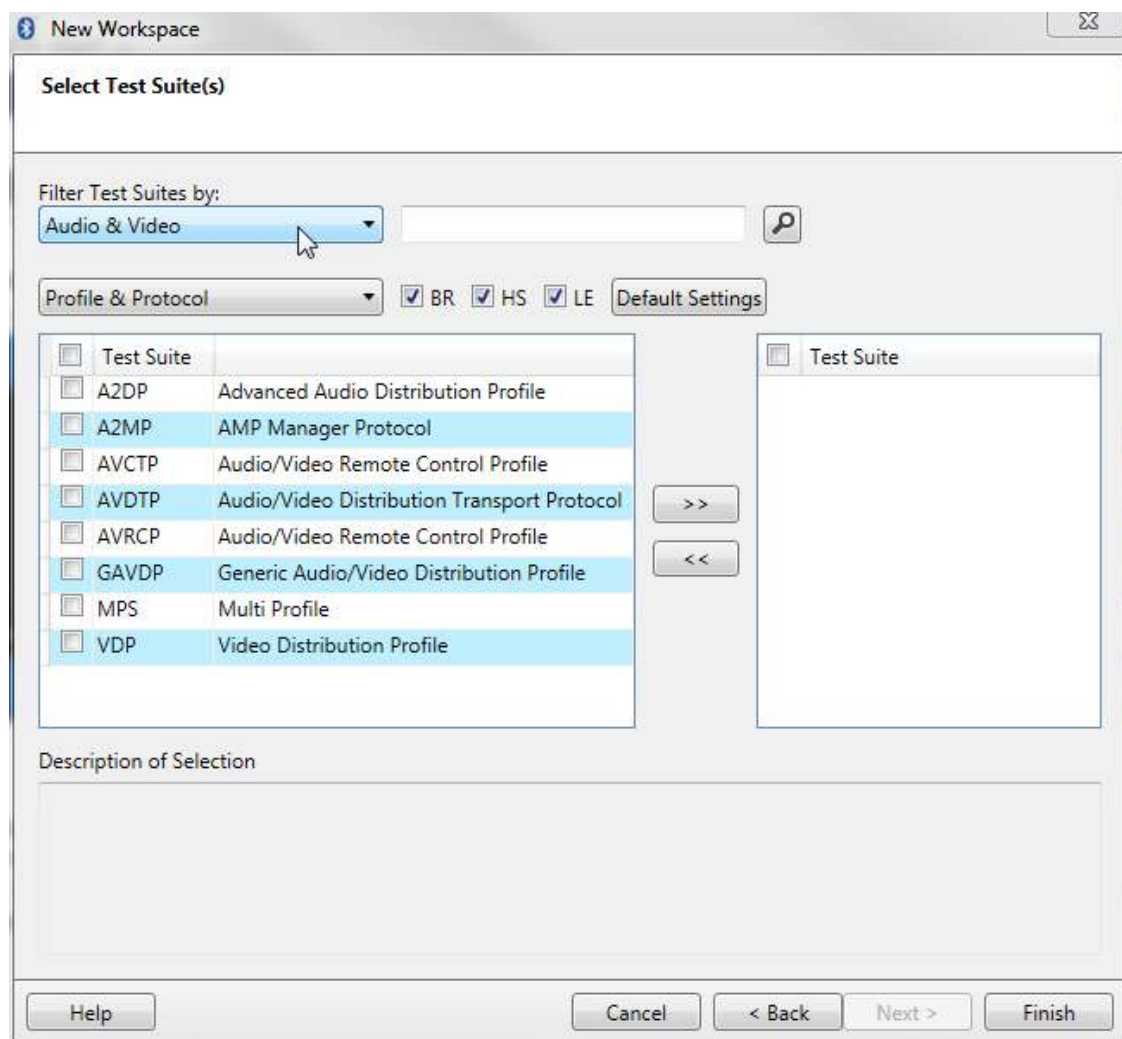
Filtering the “Test Suite(s)” window

There are four ways to reduce (filter) the number of test suites displayed in the “Test Suite(s)” window. These methods may be used in combination with each other to in order to “fine tune” the list displayed.

Filter by test group

The various test suites have been grouped together in order to quickly locate profiles and protocols that are commonly used together. The upper left hand selection box contains a list of those groups.

Click on the small arrow at the right of the list and select the group of interest.

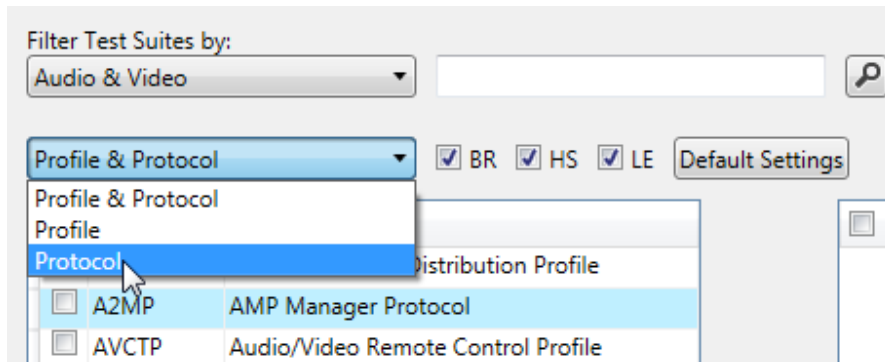


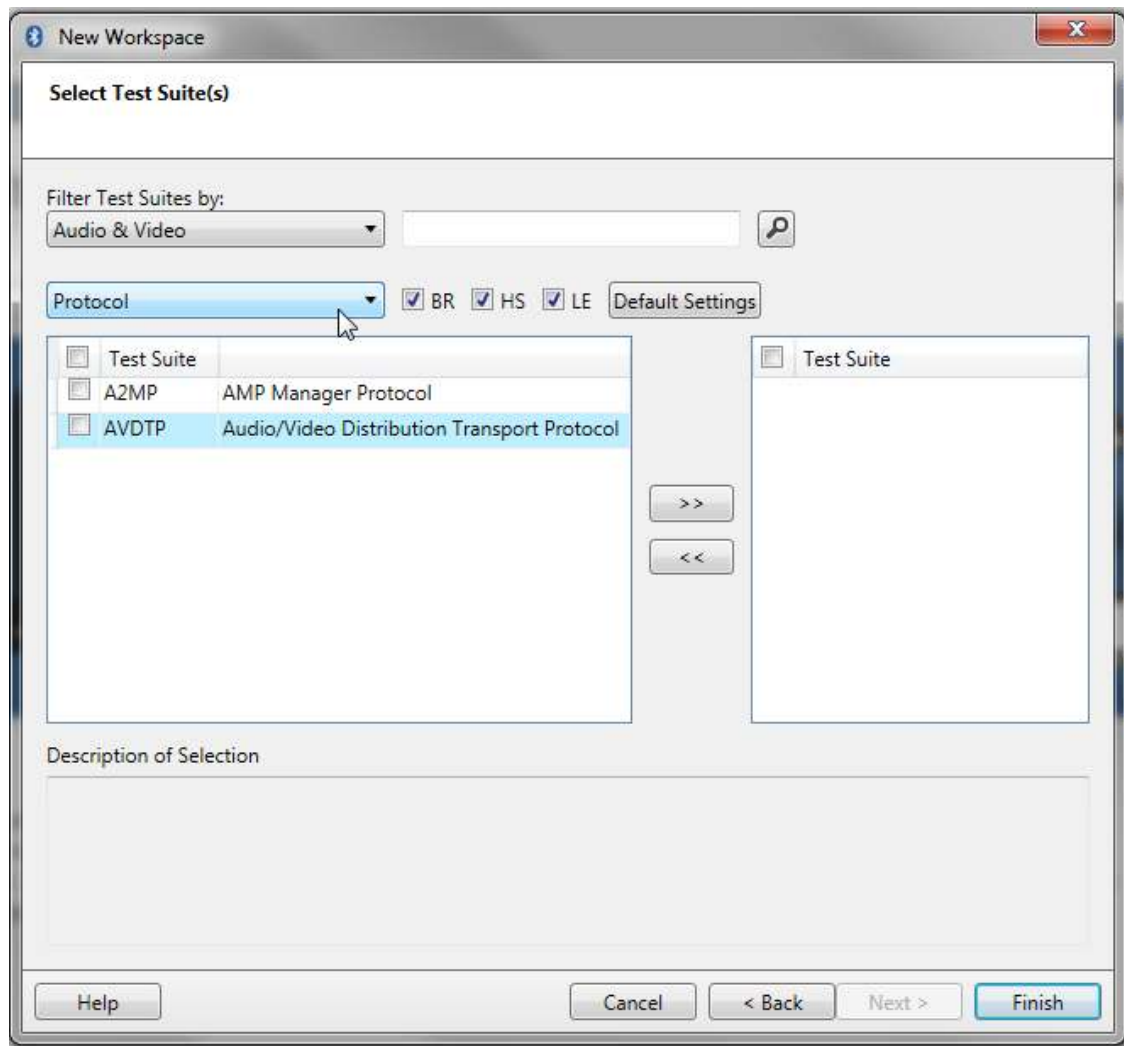
Filter by layer

PTS supports the testing of application profiles along with many of the protocol layers used to transport profile data. The “Profile & Protocol” selection box may be used to hide one or the other.

- “Profile” includes all Bluetooth application profiles along with GATT-based profiles and their associated services;
- “Protocol” includes the various transport layers such as L2CAP;
- “Profile & Protocol” includes all Bluetooth application profiles, GATT-based profiles and services, and the various transport layers.

Click on the small arrow at the right of the list and select the item of interest.





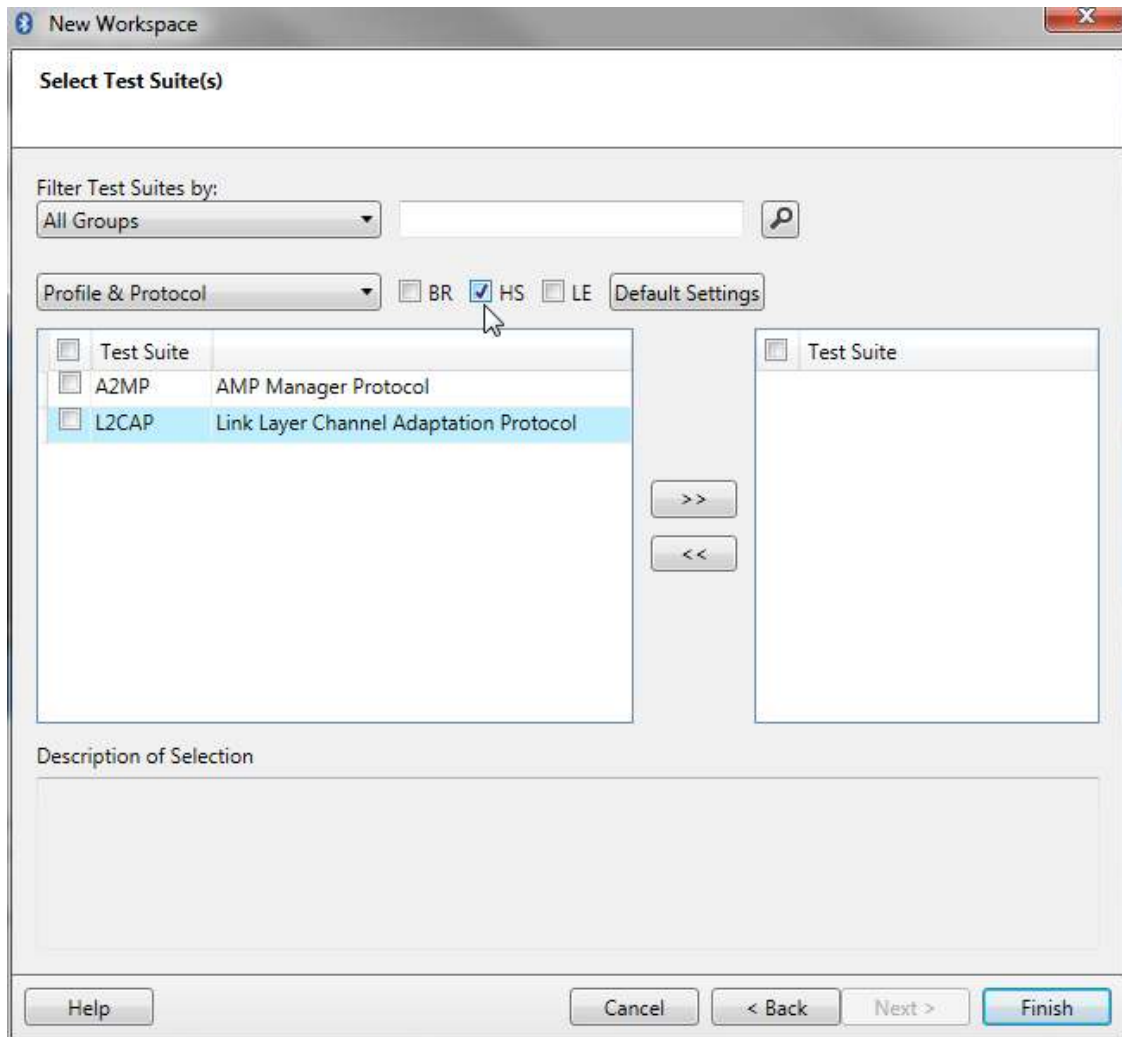
Filter by technology

Bluetooth technology currently defines three different technologies for connecting one device to another:

- “BR” – Basic Rate plus Enhanced Data Rate (often referred to as “BR/EDR”);
- “HS” – High Speed communications over a high speed data link such as IEEE 802.11;
- “LE” – Low power consumption links using Bluetooth Low Energy.

Three check boxes are present on the Test Suite Selector dialog, each representing one of the technologies.

A check in one of the boxes indicates that protocols and profiles relevant to that technology should be displayed. An empty check box will hide those protocols and profiles that are not relevant.

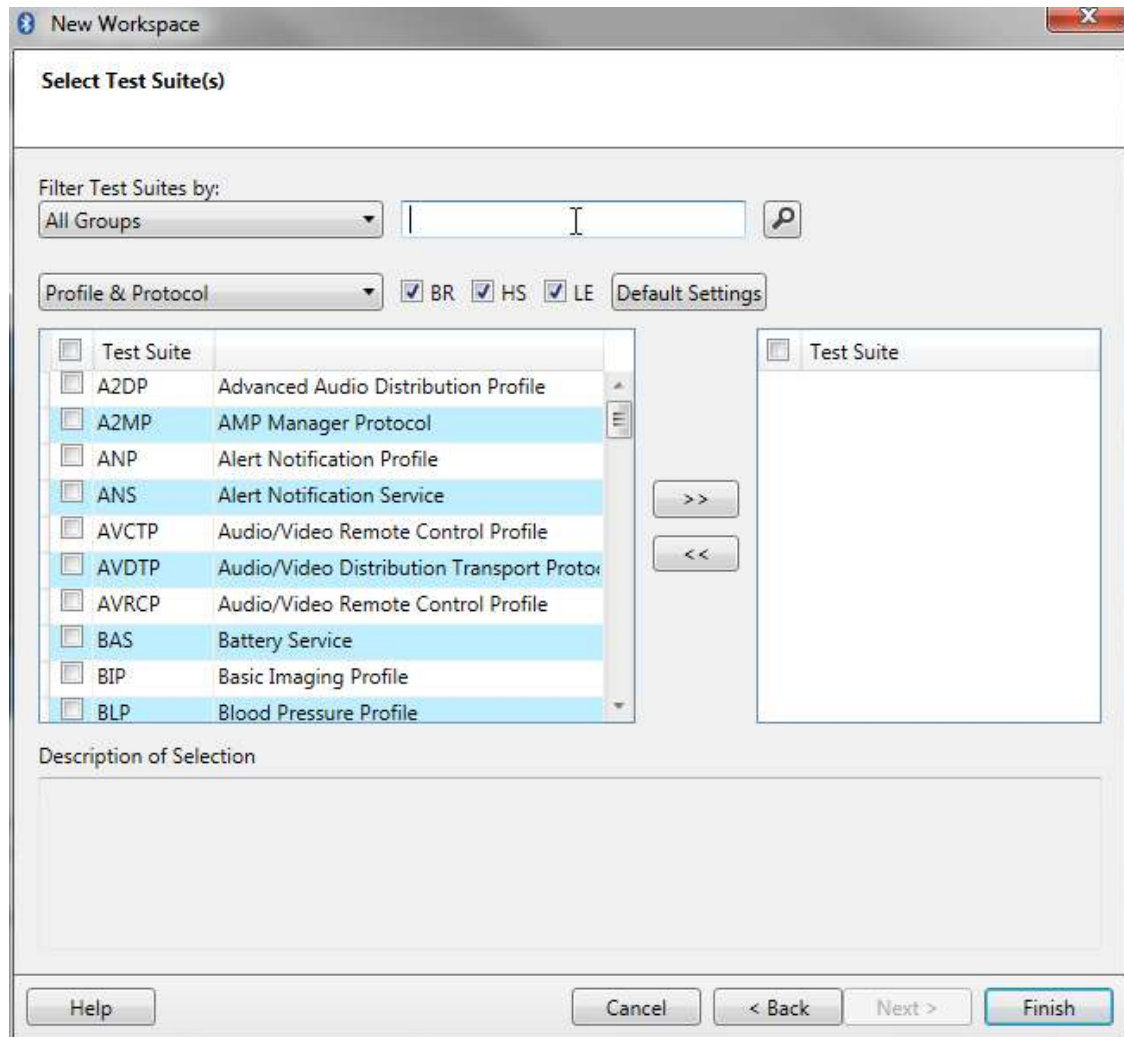


Some profiles and protocols – such as L2CAP – are used with more than one technology. This will cause them to appear in multiple technology selections.

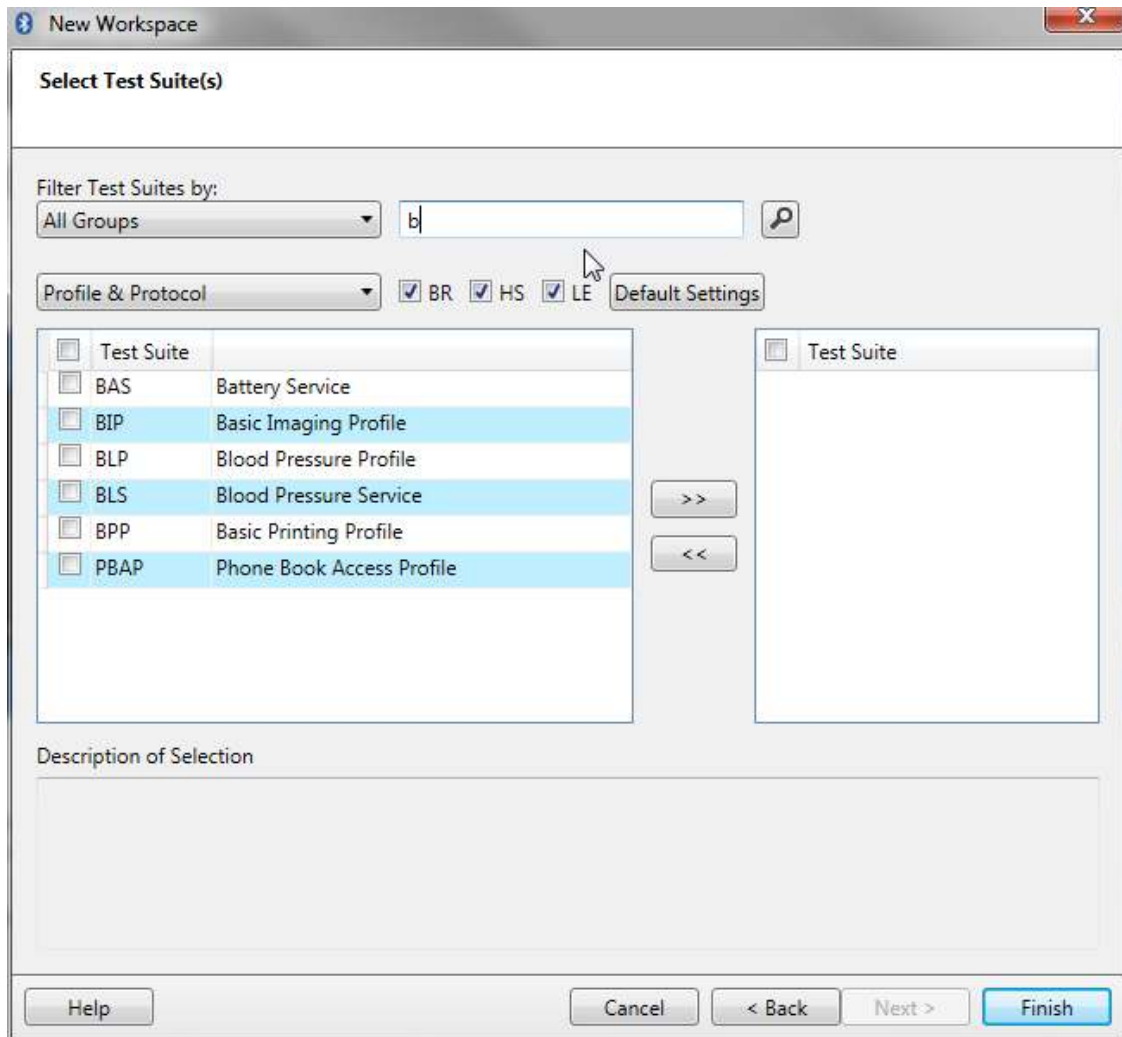
Filter by text

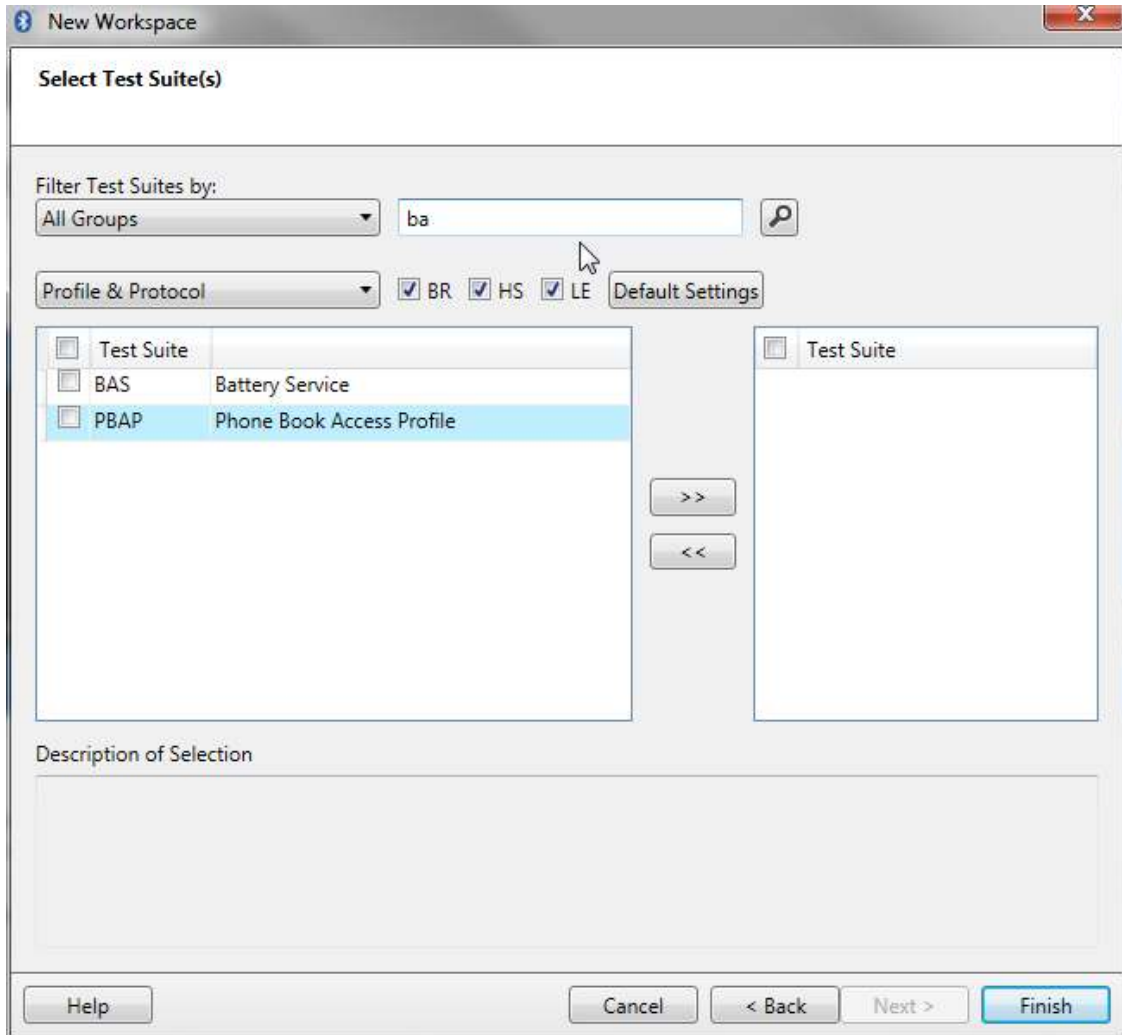
The text search filter can be used to locate test suites that contain a specific string of text in the full name or the acronym of the suite. The search is a simple search – if “acronym – name” contains the specified text it is included. The search is not case sensitive.

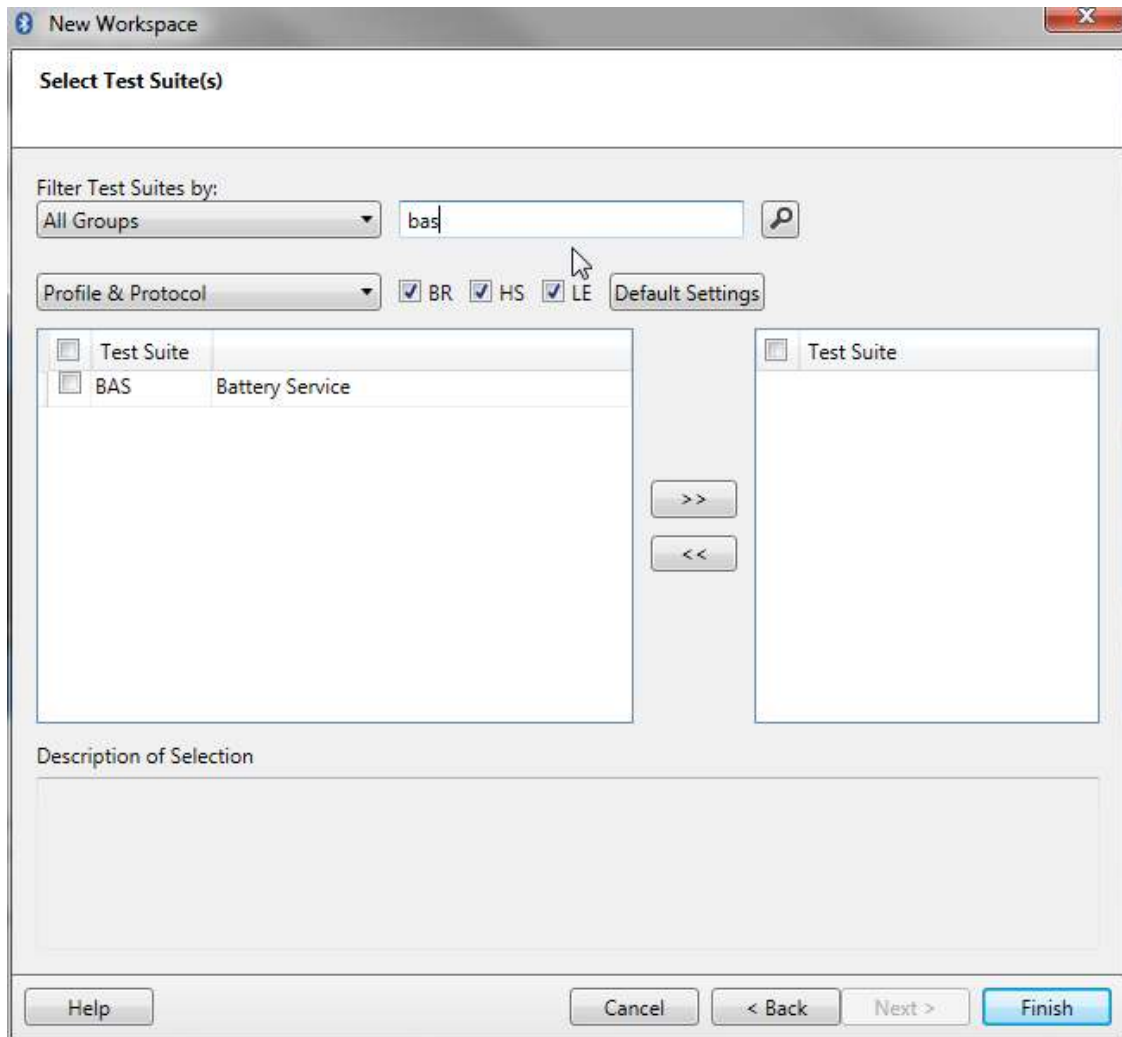
The text search filter can be used anytime the magnifying glass icon appears next to the box marked “Search...”.



The search process is dynamic and executes as text is entered or removed from the "Search..." box.







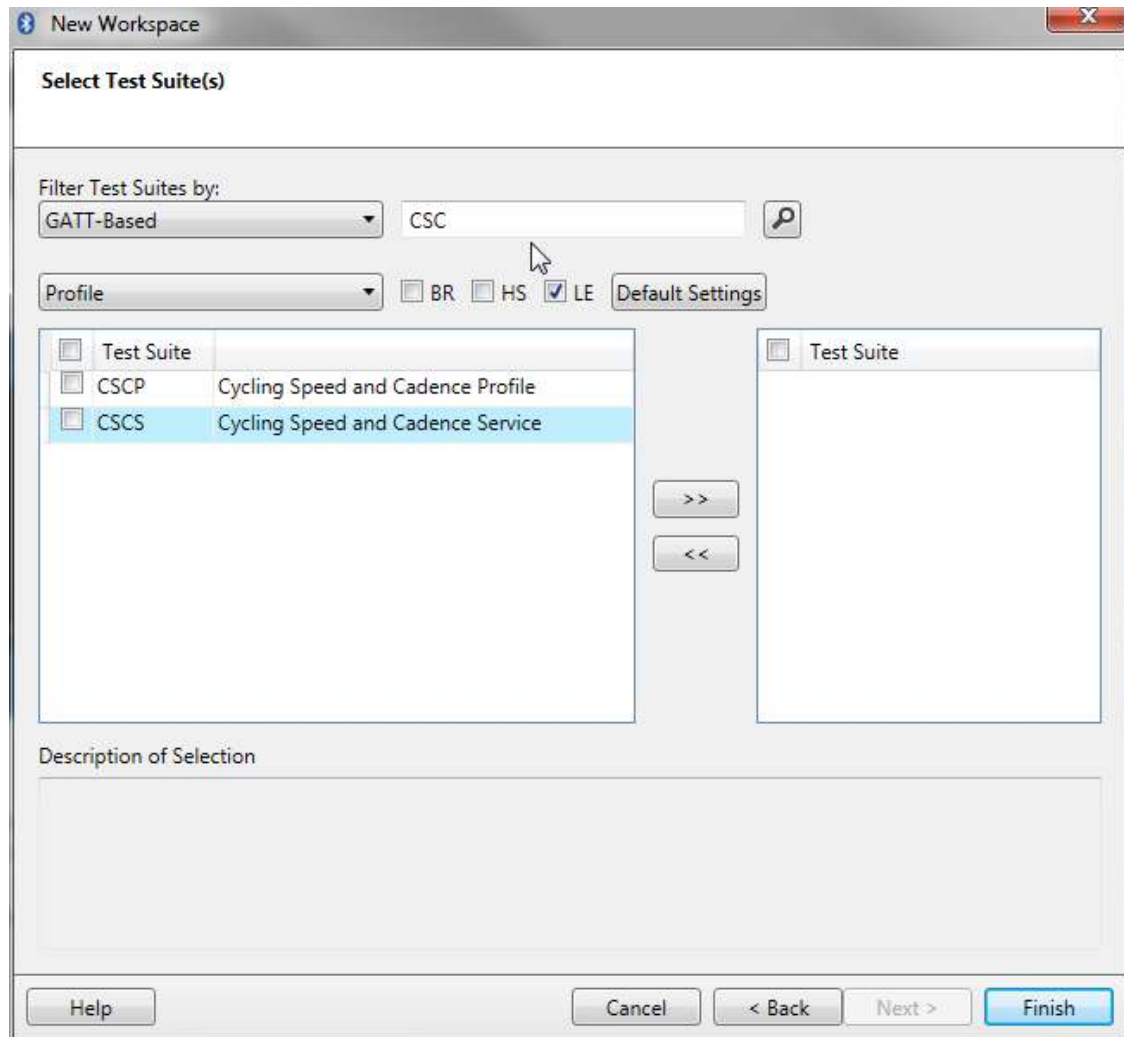
Note that as text is being entered, and the text search filter is active, the magnifying glass icon changes to a button labeled "X".

Clearing a text filter

The text search filter can be cleared by either erasing all of the characters in the "Search..." box.

Combining Filters

As mentioned earlier, the four filtering methods may be used in combination in order to arrive at a list of test suites that match multiple criteria.

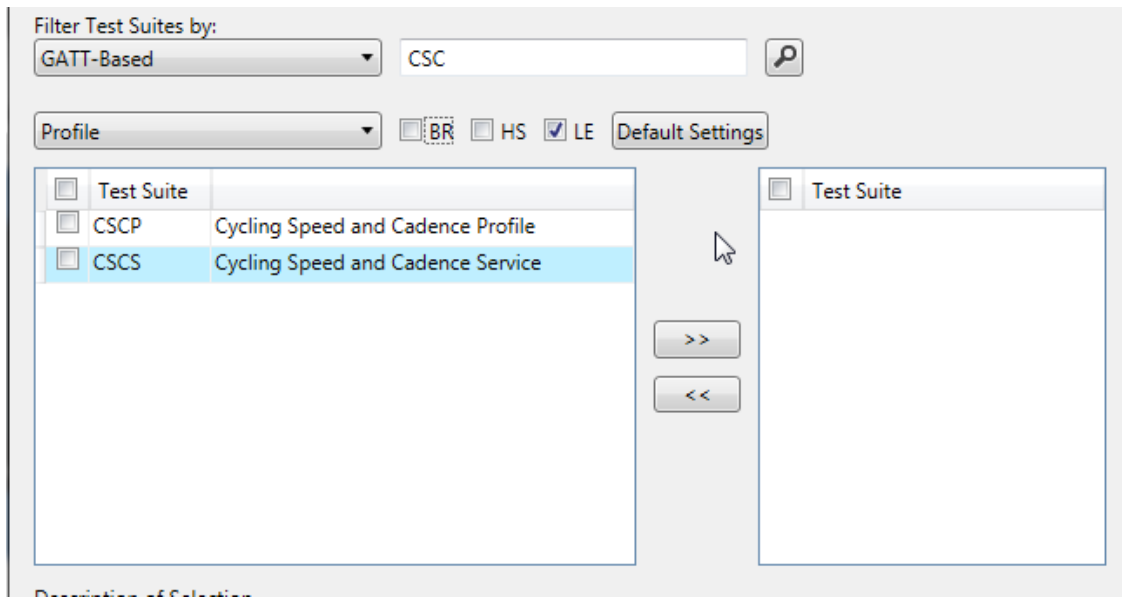


Starting over

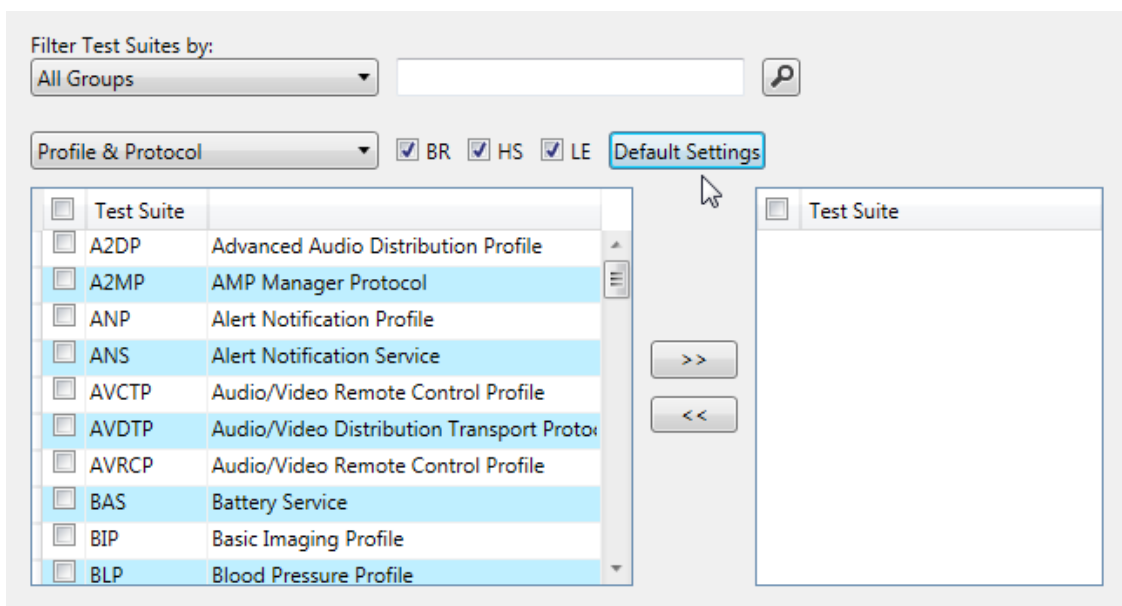
The “Default Settings” button may be used to reset the filter criteria.

- The test group filter will be reset to “All Groups”;
- The layer filter will be reset to “Profile & Protocol”;
- All three Bluetooth technologies will be selected;
- The text search string (and result) will be cleared.

Before “Default Settings”:



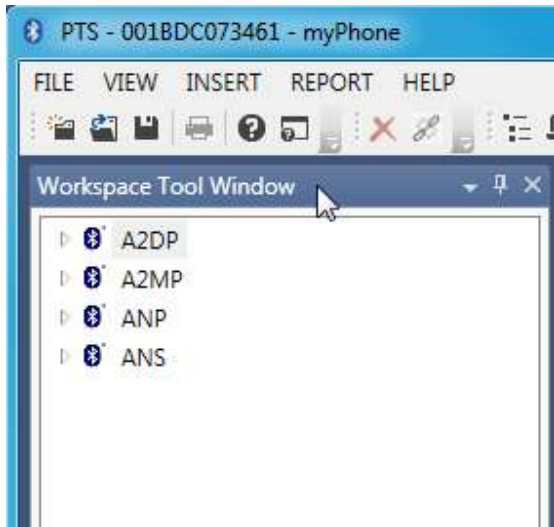
After "Default Settings":



Projects

Projects

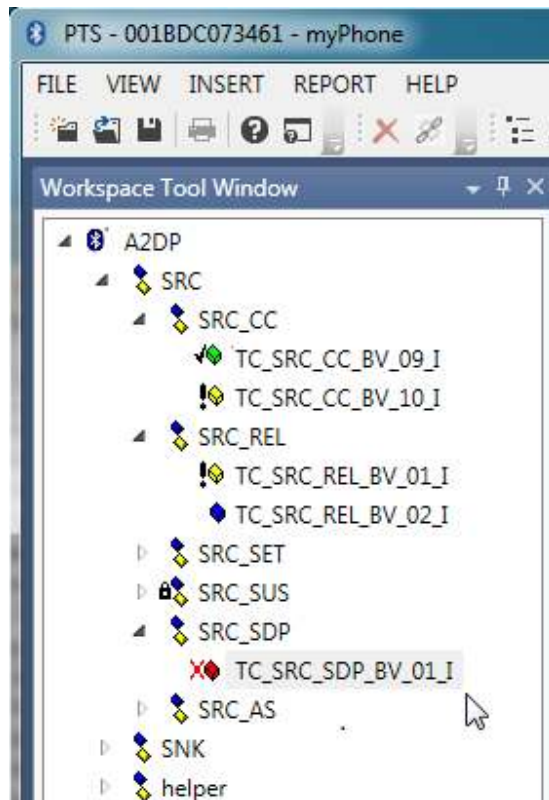
After a workspace is opened, a list of projects (profiles and protocols) that are available is displayed in the "Workspace Tool Window."



The "Test Case View" is organized as a tree where the top level items are the projects. The tree can be expanded as needed until the lowest level items – the test cases – are displayed.

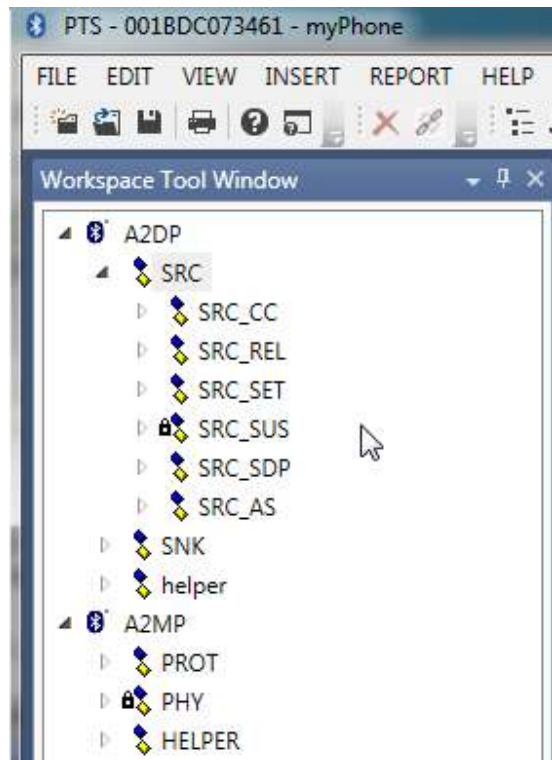
An icon next to each test case name shows the current status of that test case.

- A check mark with a green box indicates that the last run of the test case resulted in a verdict of PASS.
- A red "X" along with a red box indicates that the last run of the test case FAILED.
- An exclamation point ("!") with a yellow box shows test cases whose last run ended in a verdict of INCONCLUSIVE.
- Blue boxes with no marker in front of them are used to indicate test cases that have not been run.



Depending on the functions and features that a device supports, some test cases may not be necessary to execute in order to qualify the device. In many cases, such test cases are not likely to PASS since they exercise features that are not present in the IUT.

Test cases, or even entire groups of test cases, which are not suitable for the device to be tested are indicated by a padlock symbol next to the name of the test case or test group.



Editing the project ICS

During qualification, it is very important to be using the proper ICS settings for the Implementation Under Test. Failure to do so can result in test cases that must be executed using PTS against the IUT to be skipped.

Additionally, incorrect ICS settings can make available test cases that exercise features or functions that are not present on the device. In many cases such test cases have no chance of reaching a verdict of PASS and executing them is a waste of time.

It is also important that the ICS settings used in PTS are the same as those given in the device declaration in the Test Plan Generator. When testing evidence is submitted during qualification the list of test cases that must have been executed is based on the ICS settings in the TPG. Having different ICS settings in PTS can result in not having testing evidence for some test cases.

Generally, the best thing to do is to keep PTS in sync with the TPG by clicking the "Import ICS from TPG" feature described above. ([Creating a new workspace](#) and [Adding a project to a workspace](#))

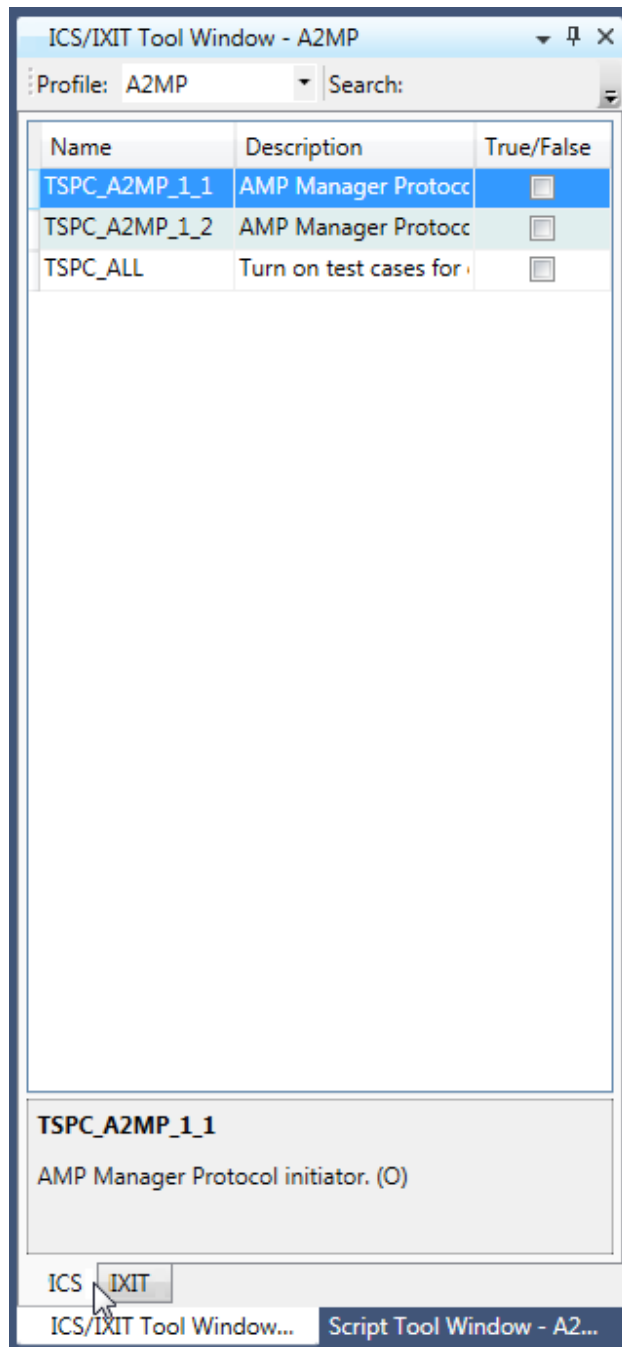
It is sometimes desirable however to change the ICS settings. This is especially true during the development of a device before the final set of features and functions have been determined.

Opening the ICS in the ICS\IXIT Tool Window

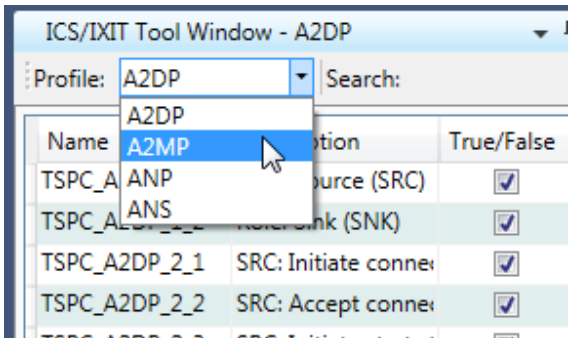
To open the ICS editor in the ICS\IXIT Tool Window, click the "ICS" button in the toolbar.



Alternatively, the ICS editor may be opened for any project in the workspace by clicking on the ICS tab on the ICS\IXIT Tool Window.



To open the ICS editor for a project, select the project from the Profile drop down menu.

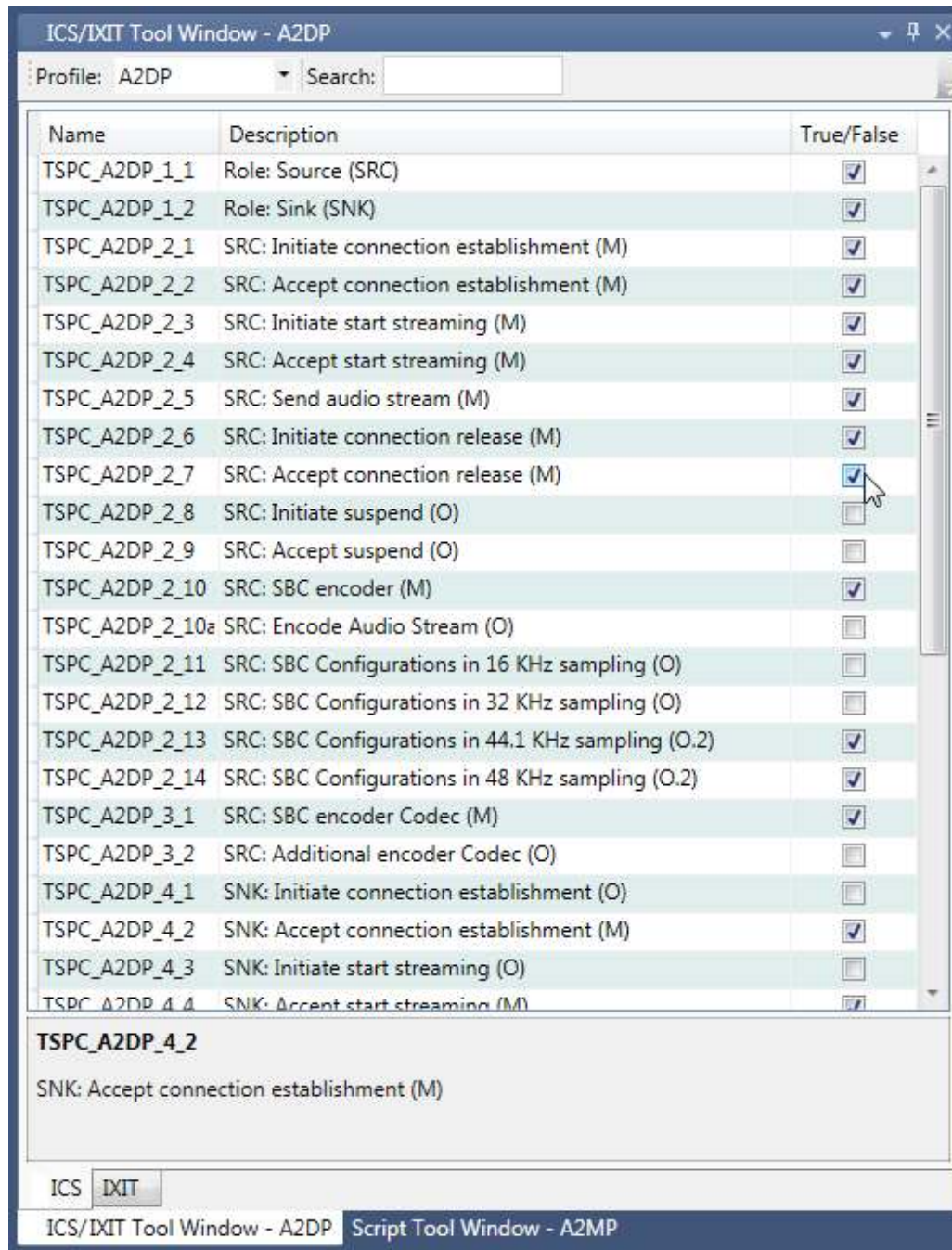


Using the ICS editor

The ICS editor displays a table consisting of three columns. Each row in the display corresponds to an item found in the ICS Proforma document that is a part of every Bluetooth profile or protocol specification.

The first column, "Name" contains the names of the ICS items. The names are based on the table and row in the ICS Proforma document where the item is defined.

For example, TSPC_HFP15_2_3 is found in table 2, row 3 of the ICS.



Proforma document for Handsfree Profile (HFP) version 1.5.

The "Description" column is used to describe the ICS items. A number of informational "cues" are used in these descriptions to make the items easier to work with.

A description ending in “(M)” represents a feature or function that a device must support. These “mandatory” items should always be selected (set to “True”).

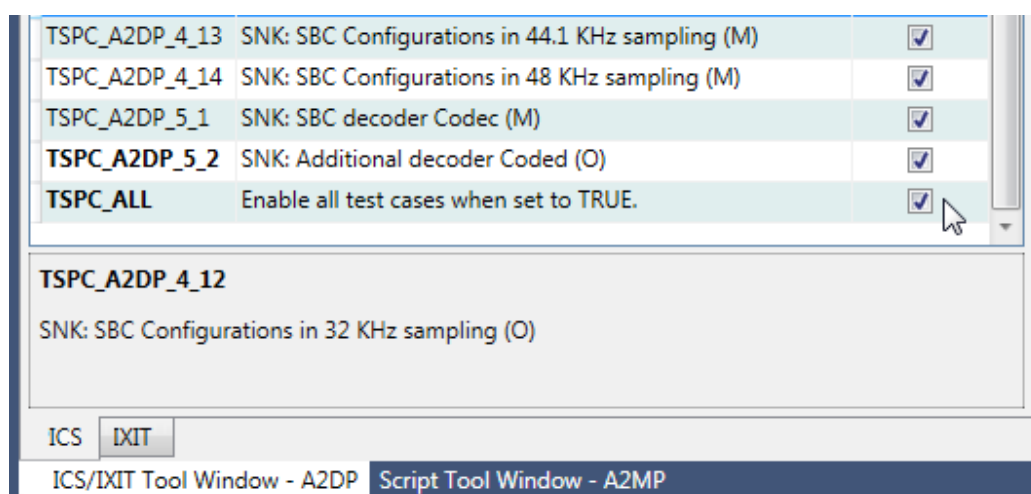
A description that ends with an “(O)” is for an optional feature or function that will be set to “True” or “False” depending on whether or not the device supports it.

The operational “role” that a ICS item applies to is often indicated at the beginning of a description. In the picture above, “AG:” is for ICS items that are applicable to a Handsfree device in the Audio Gateway (AG) role.

In the “True/False” column, check the checkbox to set the item as True, and uncheck the checkbox to set the item as False.

Enabling all test cases regardless of ICS settings

Sometimes, especially during the development of a device, it is useful to make all of the test cases available for execution. Some test cases are complements of other test cases. It may be that selecting a given ICS item enables one test case and disables another. Not selecting that item may reverse the test case selection. This often occurs in cases where it is necessary to confirm that a device responds properly when it does not support a feature or function.



The last row in the ICS editor usually contains a PTS specific item named TSPC_ALL. Selecting this item will cause all of the test cases in the project to be available to be executed.

Editing the project IXIT settings

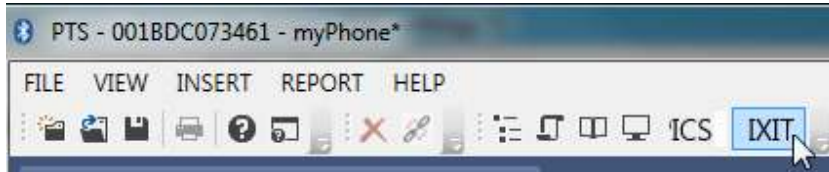
The IXIT table for a project contains entries for data elements that the test cases need in order to do their job, but that cannot be determined in advance. IXIT items are generally specific to a particular tester – such as the PTS – and are not found in any specification.

The most commonly edited IXIT item is TSPX_bd_addr_iut. This item contains the Bluetooth Device address for the device being tested. Changing this value allows a different instance of the same device to be tested.

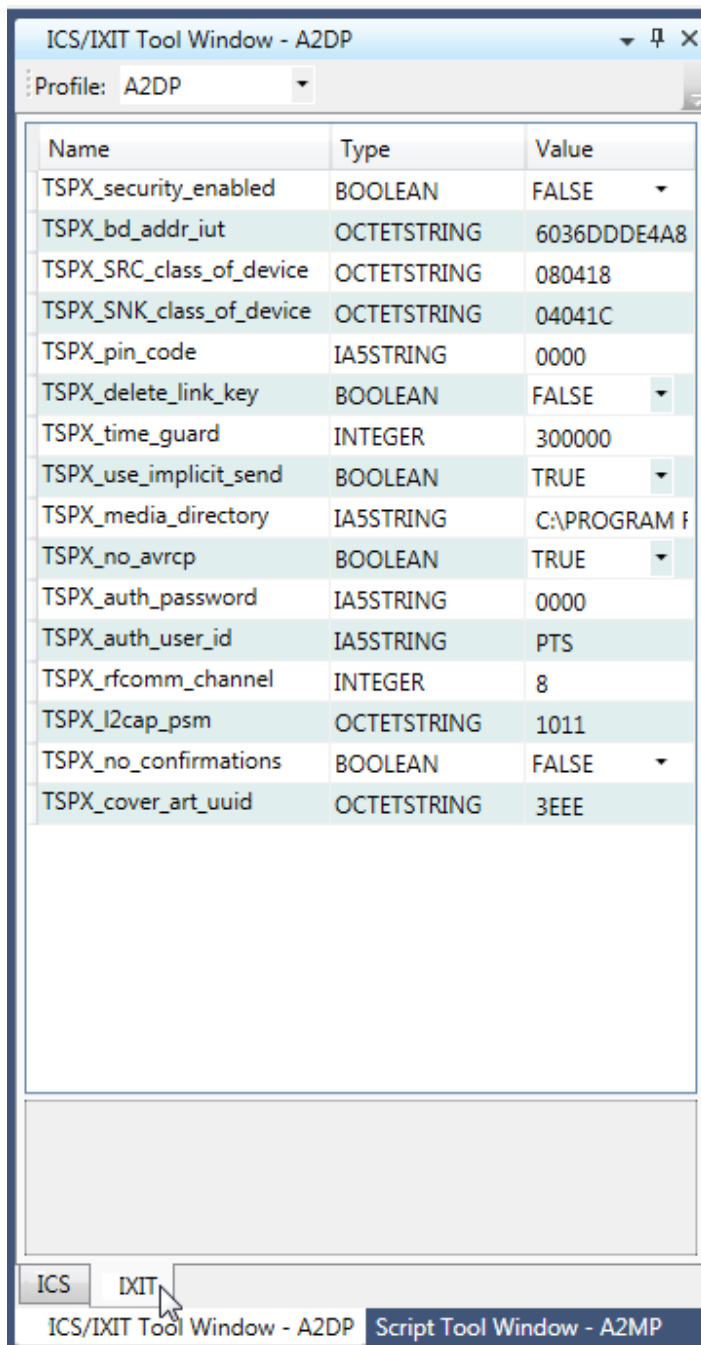
The IXIT editor works almost the same as the ICS editor, the difference being in the columns that are available.

Opening the IXIT editor

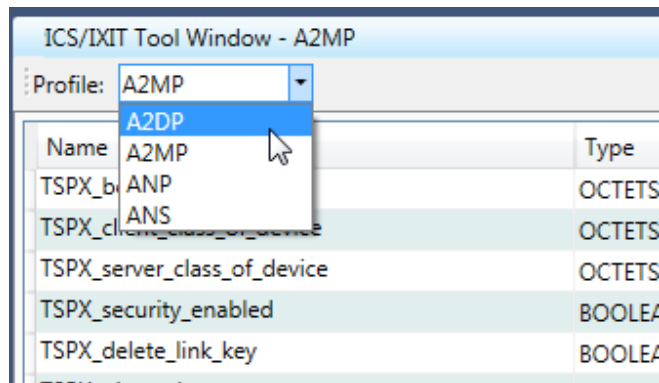
To open the IXIT editor in the ICS\IXIT Tool Window, click the "ICS" button in the toolbar.



Alternatively, the IXIT editor may be opened for any project in the workspace by clicking on the IXIT tab on the ICS\IXIT Tool Window.



To open the IXIT editor for a project, select the project from the Profile drop down menu.



Using the IXIT editor

The IXIT editor displays a table consisting of three columns.

The first column, "Parameter Name" contains the names of the IXIT items. The names should be self explanatory. (See below.)

The "Type" column is used to indicate the data type of the item that is to be entered in the "Value" column.

- "BOOLEAN" items are either "TRUE" or "FALSE".
- "INTEGER" items are decimal integer values.
- "IA5STRING" items are used for strings of textual data.
- "OCTETSTRING" items are binary values entered in hexadecimal notation.

ICS/IXIT Tool Window - A2MP

Profile: A2MP

Name	Type	Value
TSPX_bd_addr_iut	OCTETSTRING	6036DDDE4A80
TSPX_client_class_of_device	OCTETSTRING	100104
TSPX_server_class_of_device	OCTETSTRING	100104
TSPX_security_enabled	BOOLEAN	TRUE
TSPX_delete_link_key	BOOLEAN	TRUE
TSPX_pin_code	IA5STRING	0000
TSPX_flushto	OCTETSTRING	FFFF
TSPX_inmtu	OCTETSTRING	02A0
TSPX_outmtu	OCTETSTRING	02A0
TSPX_tester_role_optional	IA5STRING	L2CAP ROLE INITIATOR
TSPX_psm	OCTETSTRING	0003
TSPX_time_guard	INTEGER	180000
TSPX_timer_rtx	INTEGER	10000
TSPX_timer_rtx_max	INTEGER	60000
TSPX_timer_rtx_min	INTEGER	1000
TSPX_rfc_mode_tx_window_size	OCTETSTRING	08
TSPX_rfc_mode_max_transmit	OCTETSTRING	03
TSPX_rfc_mode_retransmission_timeout	OCTETSTRING	07D0
TSPX_rfc_mode_monitor_timeout	OCTETSTRING	2EE0
TSPX_rfc_mode_maximum_pdu_size	OCTETSTRING	02A0
TSPX_use_implicit_send	BOOLEAN	TRUE
TSPX_use_dynamic_pin	BOOLEAN	FALSE
TSPX_iut_SDII_size_in_bytes	INTEGER	144

TSPX_server_class_of_device

The indicated capabilities of PTS in server role (Default: 100104)

ICS IXIT

ICS/IXIT Tool Window - A2MP Script Tool Window - A2MP

IXIT documentation

Each test suite implemented in PTS is accompanied by an Abstract Test Suite (ATS) document that describes the test suite implementation. Among other things in an ATS document is a list of the IXIT items for that test suite along with their descriptions.

The ATS documents may be accessed from the PTS "Start Page" by selecting the "Abstract Test Suites" item.



Profile Tuning Suite (PTS)

[Home](#)
[Release Notes](#)
[Abstract Test Suite](#)
[PTS Help](#)

About Abstract Test Suites (ATS)

Abstract Test Suites define how Bluetooth enabled devices should interact with the PTS. There is an ATS document in PDF form for each Profile and Protocol included in the PTS. In ATS, you will find detailed information about each Test Case including Message Sequence Charts, mappings to the TCRL, IXTT information and test purposes.

ATS documents for profile and protocol test suites:

A2DP	DI	HCRP	MAP	SAP
A2MP	DUN	HDP	MCAP	SM
AVCTP	FTP	HFP	MPS	SPP
AVDTP	GAP	HD	OPP	SYNC
AVRCP	GATT	HSP	PAN	VDP
BP	GAVDP	IOPT	PBAP	
BPP	GNSS	L2CAP	RFCOMM	

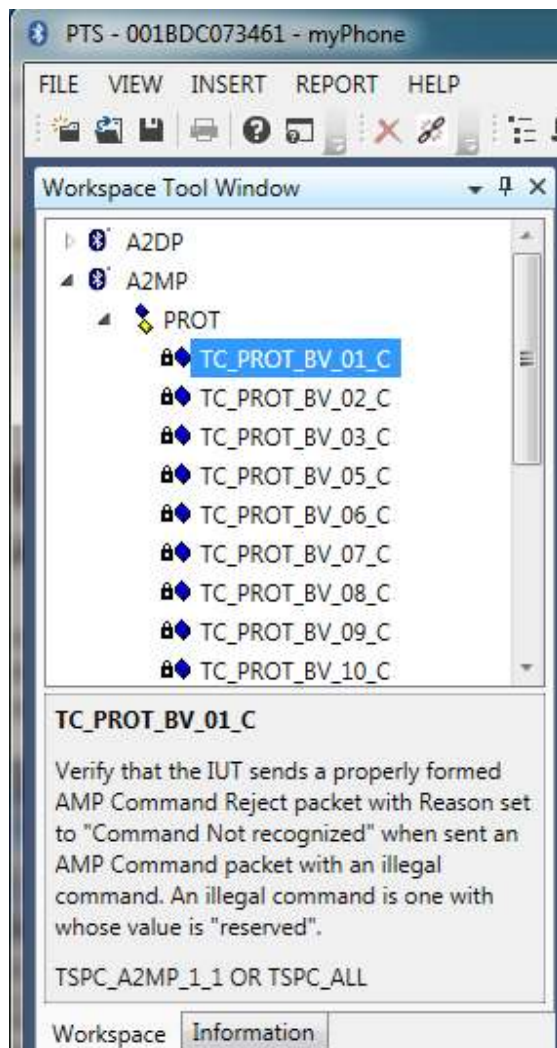
ATS documents for GATT-based profile and service test suites:

ANP	CSCS	HOGP	LNS	RSCS
ANS	CTS	HRP	LLS	RTUS
BAS	DIS	HRS	MODS	ScPP
BLP	FMP	HTP	PASP	ScPS
BLS	GLP	HTS	PASS	TIP
CPP	GLS	IAS	PXP	TPS
CPS	HDS	LMP	RSCP	UDS
CSCP				

Test Cases

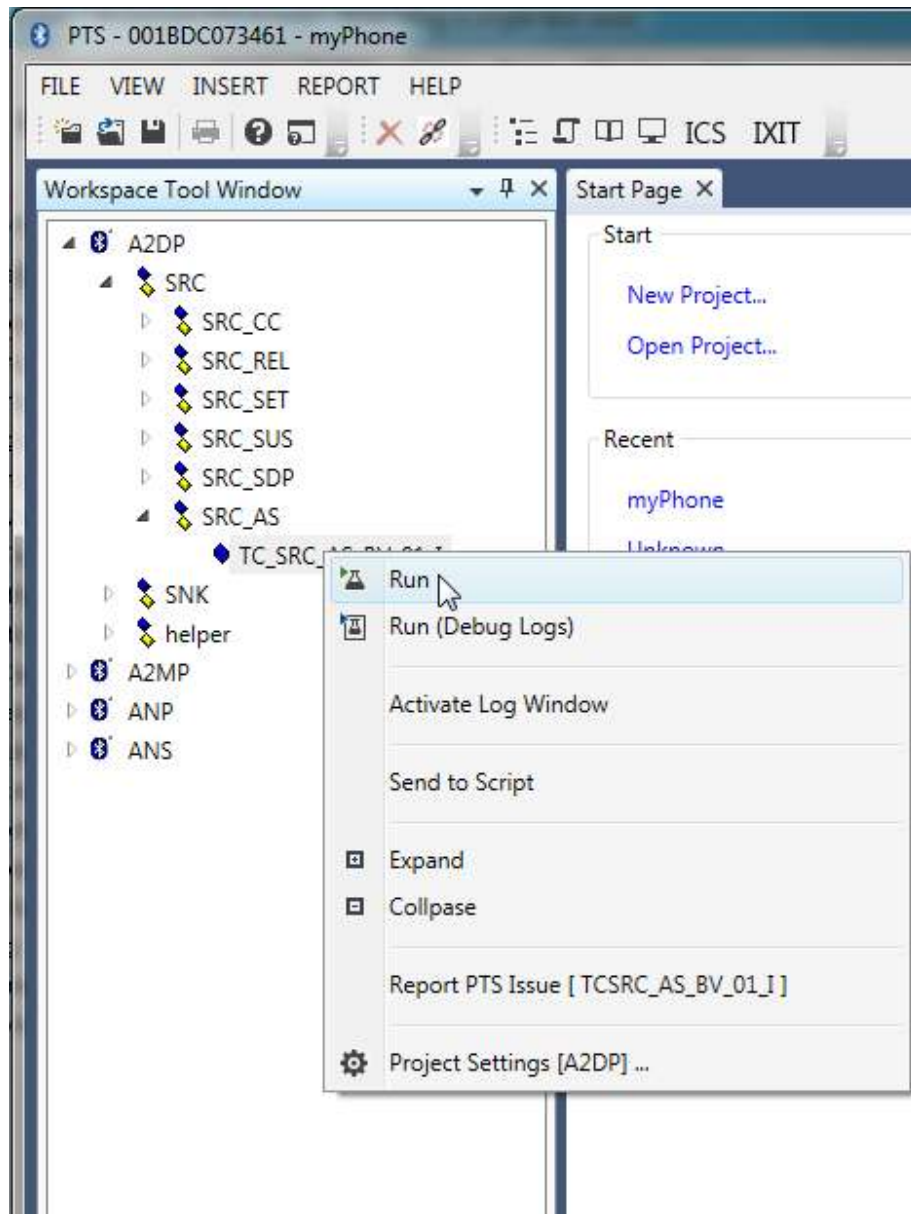
Displaying the purpose of a test case

Each test case described in a profile or protocol test specification begins with a short summary stating its purpose. For convenience, the purpose the selected test is displayed in the bottom pane of the Workspace Tool Window.

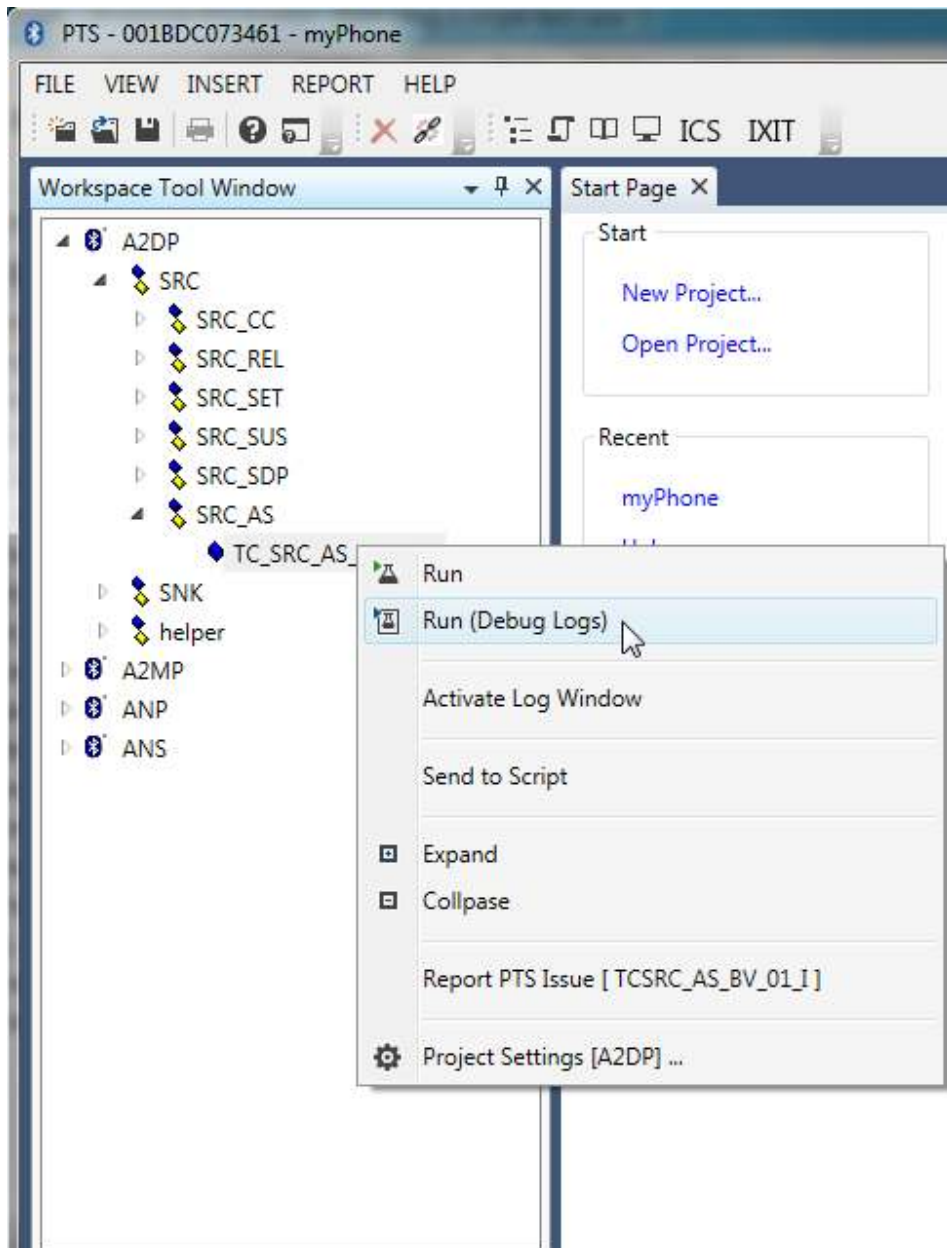


Executing a single test case

There are three ways to start the execution of a particular test case. The first way is to right click on the name of a test case and select "Run" from the popup menu.



The second method is also to right click on the name of a test case, but instead select 'Run (Debug Logs)' from the popup menu.



The final method is to locate the name of the test case in the tree and double click on its name.

Run” versus “Run (Debug Logs)”

Test cases produce a number of output logs as they are executing. The amount of information in each log can vary depending on the logging level and other settings. (See [Logging](#).)

Generally the “Standard” logging level is sufficient for most users. That level will log basic communications between PTS and the Implementation Under Test, along with various status messages at key points during the test execution.

Sometimes, the “Standard” logging level does not contain sufficient information to diagnose a problem with either PTS or the IUT. Before version 4.5.3 of PTS, it was necessary to manually set “Full” logging and to locate the additional log files that are used in tracking down an issue. For some of the log files, manually editing the program configuration file was needed to achieve the most comprehensive output.

After collecting all of the relevant information, the operator would then want to set the logging level back to “Standard” and undo the changes made to the configuration file.

“Run (Debug Logs)” simplifies the process of gathering all of the relevant logs by placing the test case execution log, and other important log files in a folder after a test case executes. This mode also temporarily overrides all of the logging settings and sets them to their maximum (most comprehensive) values.

After the test case has finished execution, the logging settings are automatically restored to their previous values.

It should be noted that when reporting a problem for particular test case to the PTS Development Team, “Run (Debug Logs)” should be used and the contents of the resulting folder should be attached to the problem report.

Aborting test case execution

When a test case is running a red “X” will be enabled in the toolbar. Clicking this button will cause the current test case execution to terminate.

Note that this only terminates the currently executing test case. If a group of test cases are being executed – as described in the previous section – the next case will be started.

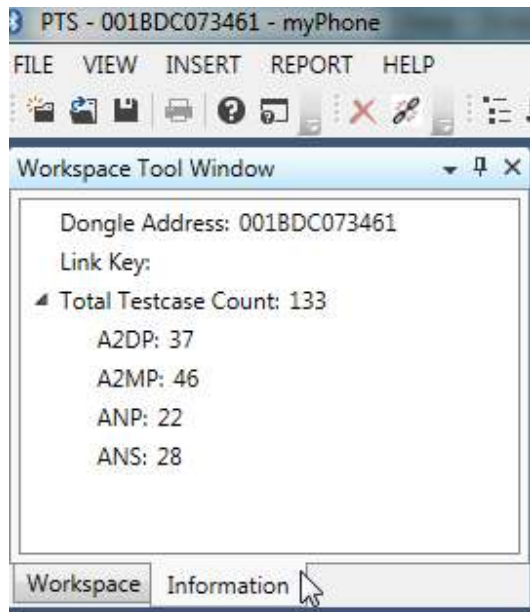


Link Keys and PTS Endpoint Information

Endpoint information

The Workspace Tool Window contains two tabs. The first tab, labeled “Workspace”, is the view of the current workspace that has been shown throughout this document.

The second tab – “Information” – contains information about the PTS Endpoint device. When no workspace is open, the “Information” section lists the Bluetooth Device Address (BD_ADDR) of the PTS Endpoint. This can be very useful when you need to locate PTS using its BD_ADDR.



When a workspace is active, the "Information" section will still list the Endpoint address, but it will also list the Link Key that is shared between PTS and the IUT if the devices have been bonded. This information is important for two reasons:

1. It indicates that PTS thinks that there is a trusted relationship between itself and the IUT. The absence of a Link Key shows that the two devices are not bonded.
2. "Air sniffing" protocol analyzers often need the current Link Key to be entered manually in order to decrypt the data flowing between the two devices. The value shown in the "Information" section can be entered into the "Air Sniffer" to allow packets to be decrypted successfully.

Deleting the current Link Key

At times it may be necessary to remove the trusted (bonded) relationship between the PTS and the IUT. The "Delete Link Key" button on the PTS toolbar can be used for this purpose.



Once the "Delete Link Key" button has been pressed, the devices are no longer bonded.

PTS Program Settings

PTS Program Settings

There are two sets of user settings in the Profile Tuning Suite. The first set – referred to as “application settings” – affects the main PTS application.

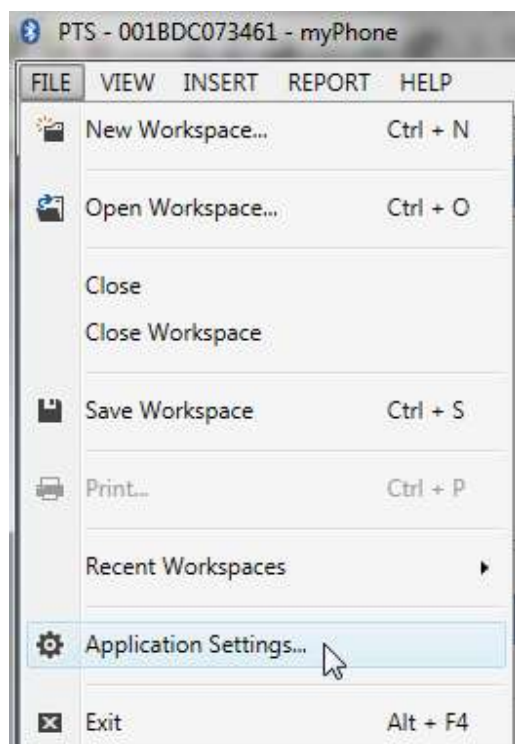
The other group of settings affects the operation of the test suites and is referred to as “project settings”.

Application settings

There are two sets of user settings in the Profile Tuning Suite. The first set – referred to as “application settings” – affects the main PTS application.

The other group of settings affects the operation of the test suites and is referred to as “project settings”.

The application settings are accessed using the “Application Settings” item on the “File” menu.



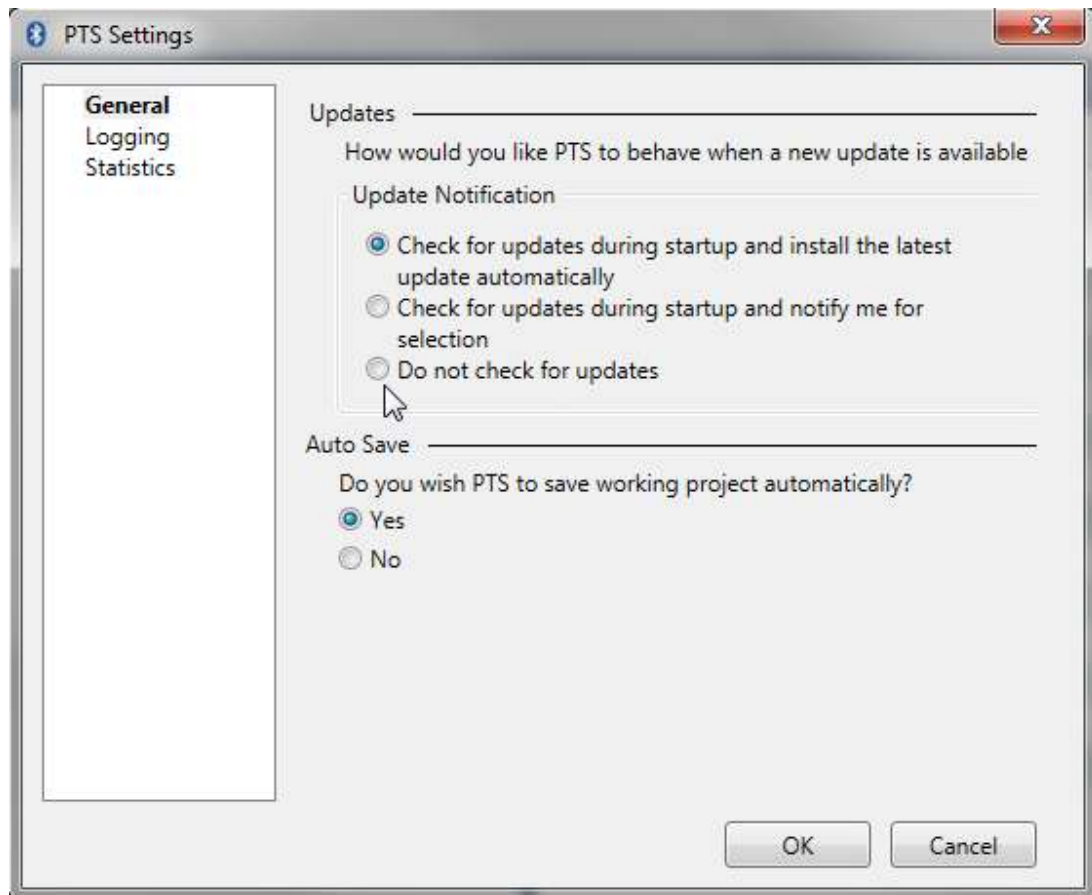
General

Updates

PTS can be configured to check for updated software whenever it is launched. You can also set PTS to notify you if updates are available. The “Updates” section in the General Application Settings tab is used to enable or disable this functionality.

Auto save

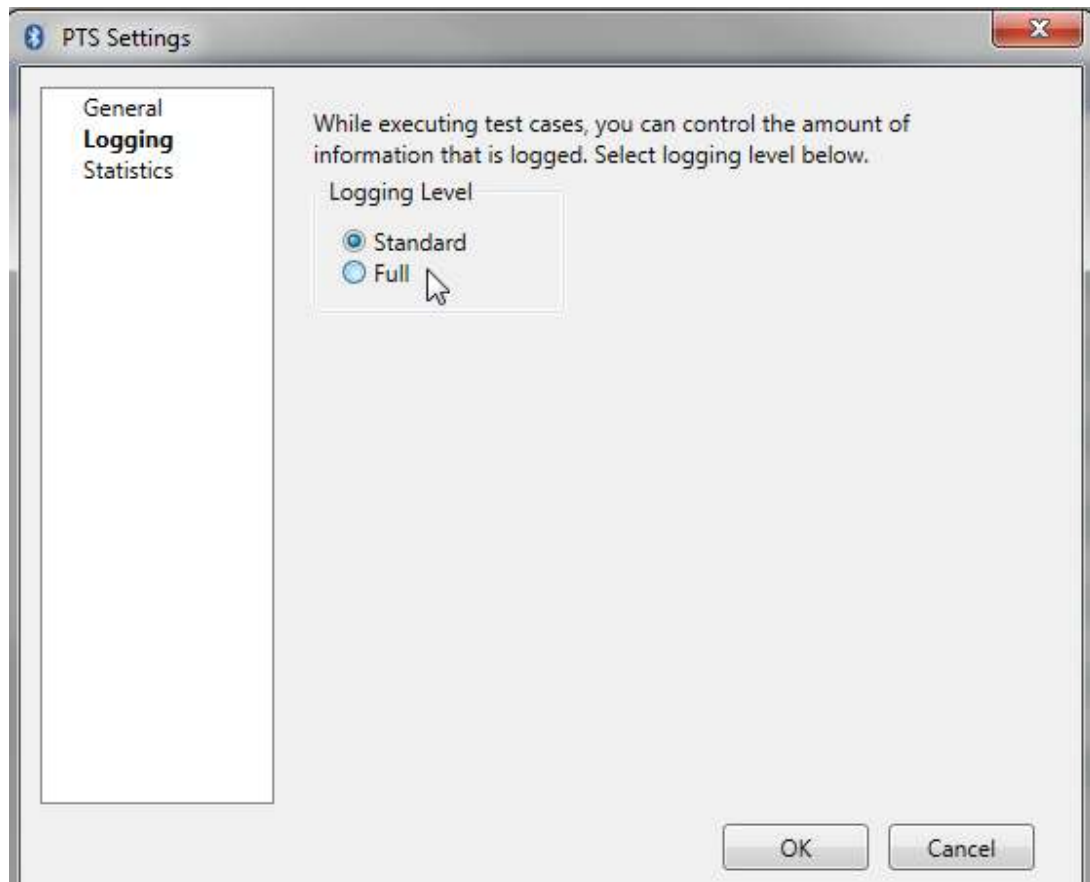
You set PTS to save a project automatically, or to only save manually.



Logging

The "Logging" option is used to control the amount of detail that is present in the execution log. The amount of detail can be selected from very little information up to highly detailed information about the inner workings of test cases.

- The "Standard" setting adds communication messages and information used to determine the final verdict of the test case to the information displayed when "Minimal" is selected.
- "Full" causes highly detailed information about the inner workings of a test case to be included in the execution log. This is a lot of information and may be overwhelming to most users.



As mentioned in [Executing a single test case](#) ("Run" versus "Run (Debug Logs)"), changes to the log level may never be needed. The "Run (Debug Logs)" feature is a much better way to produce highly detailed logs for problem analysis or archival purposes.

For more information on the various logging capabilities of PTS, please refer to [Logging](#).

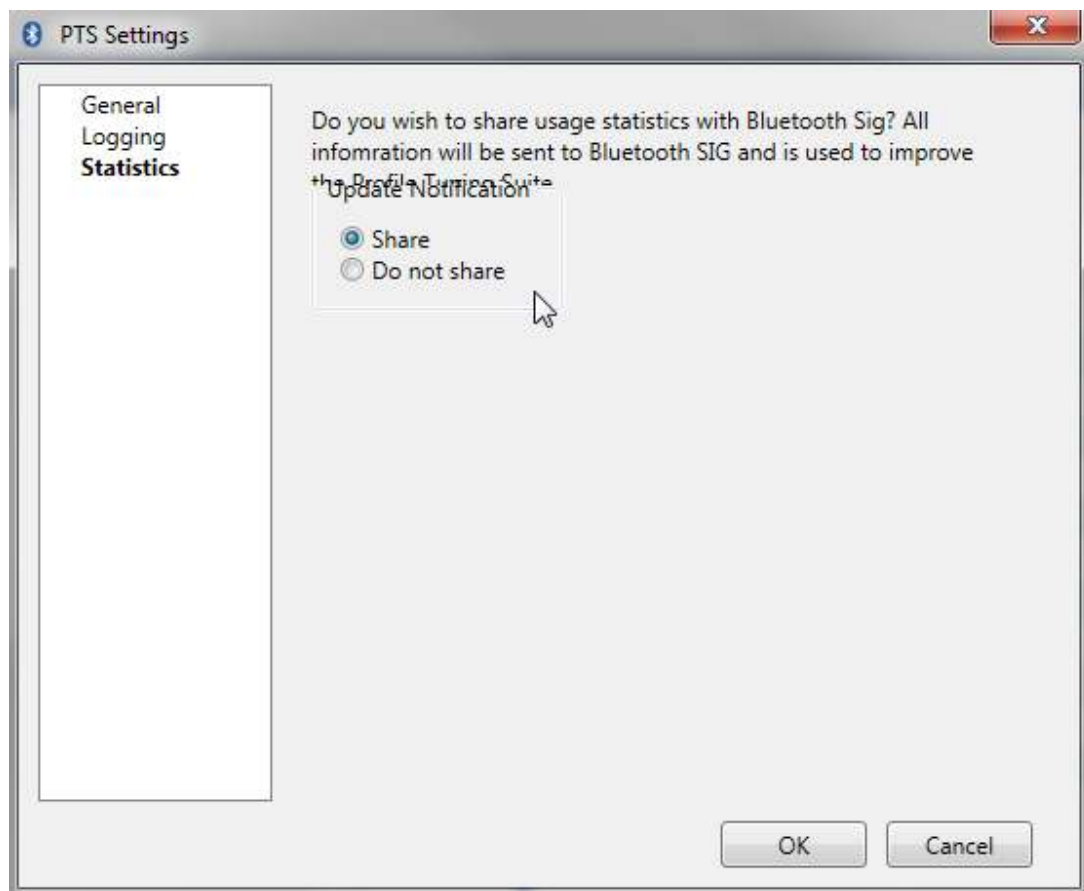
Statistics

case is executed. This information can be sent to the PTS Development Team on a periodic basis to help determine which test suites and test cases are being used the most.

This information is always sent anonymously.

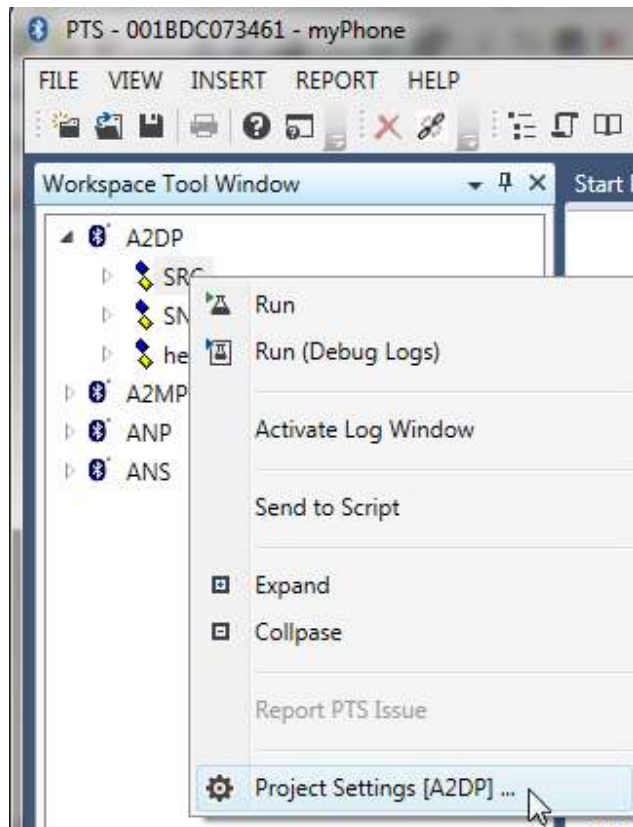
There are three settings which determine how the information is sent to the Development Team:

- Share silently": Send the information on a periodic basis without prompting for permission to do so.
- Prompt before sharing": Send the information on a periodic basis, but ask for permission before doing so.
- "Do not share": Never send the usage information automatically.



Project Settings

The project settings are accessed by right clicking on a project name at the top level of the "Test Case View" followed by the selection of "Settings..." from the popup menu.



General settings

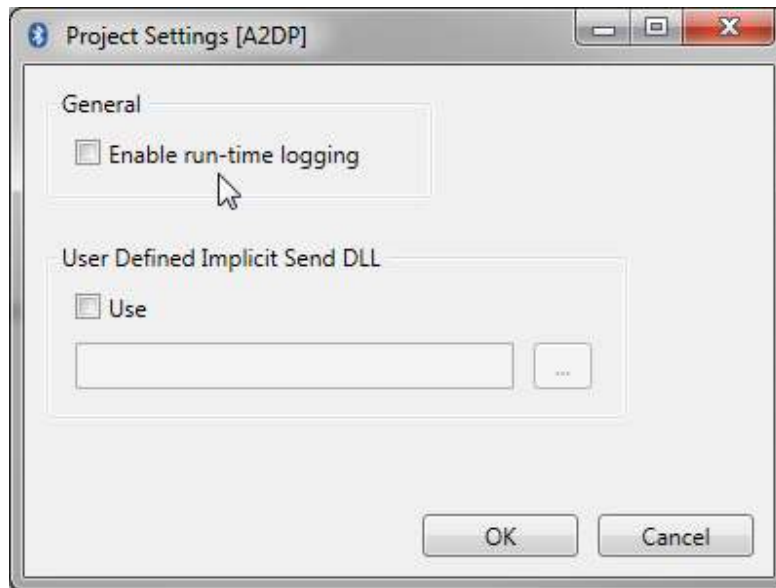
The settings on this tab control the type of output produced by the test suites and the amount of detail that is included. The selections on this tab are global and affect all projects in the workspace.

The “Enable run-time logging” option determines whether or not the log is created while a test case is executing. Sometimes, in cases where the creation of the log is impacting the performance of the test case, it may be useful to delay the creation of the log until after a test case has ended.

User Defined Implicit Send DLL

An alternate Windows DLL may be used to handle “implicit send” messages that occur during a test case. The use of alternate “Implicit Send DLLs” is uncommon, so this item should generally be left unchecked.

Users needing to do automated testing may find that the creation of a custom Implicit Send DLL is needed. Please see the “Automating Test Execution” reference document for more information.



Automating

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.
2. This feature can be used with either of the following program control features and is described in this document.
3. "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.
4. The "Scripting" reference document describes this feature.
5. "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.

"Fully Automated Operation" is described in the "Extended Automating" reference document.

"Operator-less Operation"

Many, if not most, of the Bluetooth qualification tests are designed around the idea that a human test operator is part of test environment – operating the tester and performing manual operations on the Implementation Under Test (IUT). This can be seen in the various test specifications in comments like

- Expected Outcome

Pass verdict:

The Object Push operation is processed correctly and completed corresponding to the settings and user actions.

Client:

- The Object Push function is initiated by user action and not automatically.

Part of the reason for this is that the Bluetooth Special Interest Group (SIG) does not specify the ways in which users are to interact with Bluetooth enabled devices. This can also be seen in the test specifications

- Test Procedure

...

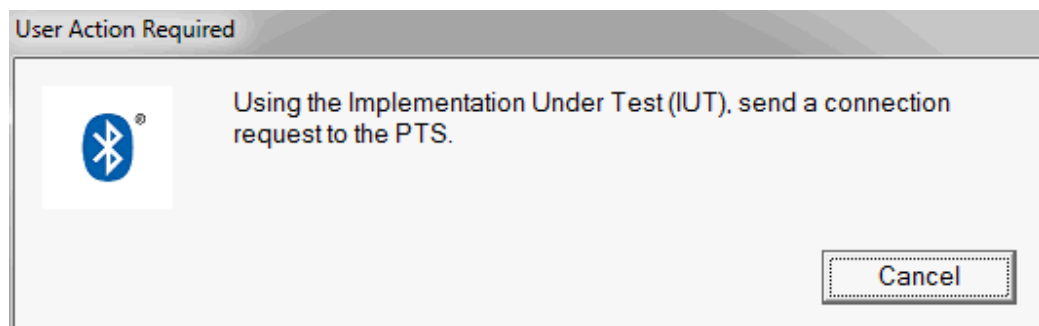
Server:

...

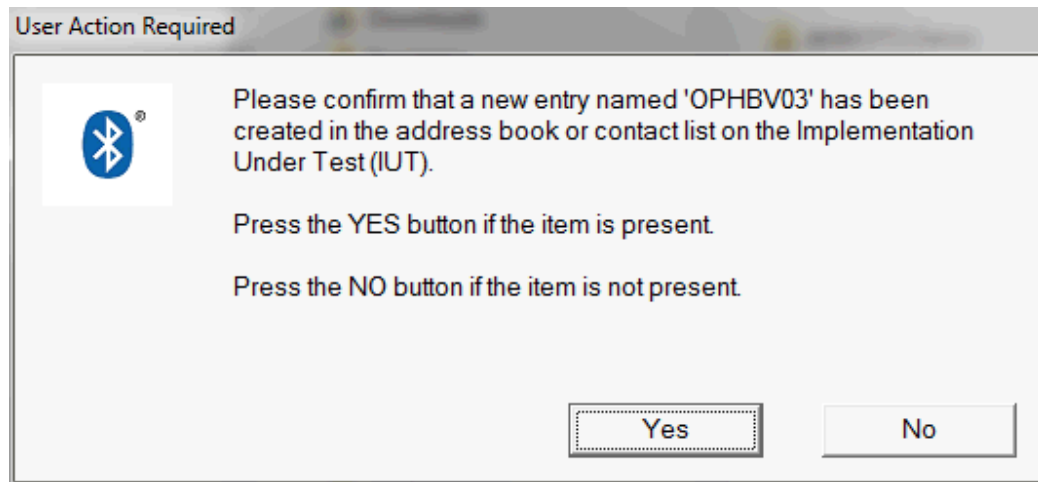
- *(Depending on the architecture that is to use the object push feature the steps how an item is pushed may vary*

In its default configuration, the Profile Tuning Suite (PTS) presents the test operator with various popup dialogs during the execution of a test case. These dialogs may be used to

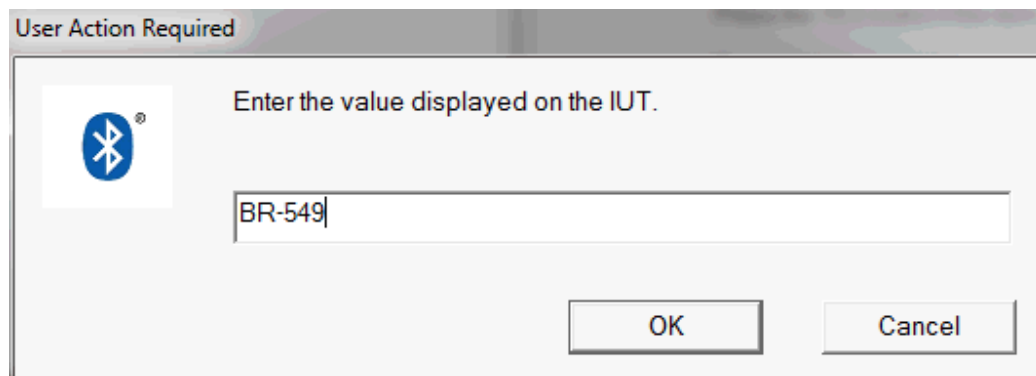
- Ask the test operator to perform a function on the IUT;



- Ask the test operator to confirm that a file transfer or other operation completed successfully;



- Ask the test operator to enter data needed for the test.



Automation test platforms

Having a test operator involved in the testing process can be very time consuming. Additionally, regression testing becomes somewhat difficult since a test operator needs to "babysit" the testing process. For this reason, many organizations create automation test platforms to be used in the testing of their devices. These platforms may have the ability to press buttons, recognize prompts and messages on the display, or access the storage on the device for contact items, pictures, or other files. These operations are controlled by software running on a computer that is connected to the test platform.

PTS has the ability to work with automation test platforms by providing a user defined mechanism that can be used to replace the popup dialogs mentioned above. Instead of sending the various messages to the display, PTS can be configured to send them to user written replacement functions that in turn can interact directly with the device being tested.

PTS test case operation

As mentioned above, there are various points during test case execution where PTS needs to interact with the test operator or an automation platform. When this occurs

1. The part of the test case that needs outside assistance sends a request to the “MMI Handler”. PTS test cases are implemented using a Main Test Component (MTC) and some number of Parallel Test Components (PTCs). The PTCs provide various support functions and operate concurrently with the main body of the test. PTCs are often used to implement a protocol layer in the Bluetooth stack or to serve as the “knowledgeable authority” for the details of a Bluetooth profile.

The MTC and the various PTCs interact with one another to process commands, responses and the transfer of data between themselves and the Implementation Under Test (IUT).

The MMI PTC handles the interaction with the outside environment. Since it operates in parallel with the other parts of the test, test case execution is not held up while waiting for a response from the test operator.

2. After receiving a request, the MMI PTC passes it on to a support library known as an “Implicit Send DLL”.
3. The Implicit Send DLL performs whatever steps are needed to execute the request and waits for a response.
4. The response is sent back up the chain to the MMI PTC, and from there to whatever Test Component is expecting it.

Implicit Send DLLs

Basic Information

PTS provides a default version of the Implicit Send DLL. It is this DLL that provides the popup dialogs that one normally sees when using the PTS.

In order to integrate PTS with an automation test platform, a custom Implicit Send DLL needs to be developed.

- Implicit Send DLLs are standard Windows Dynamic Link Libraries.
- Implicit Send DLLs are written in C++.

The interface between the PTS and an Implicit Send DLL uses the `std::string` class from the Standard Template Library (STL), and the “bool” data type. No other C++ features are used, so someone knowledgeable in the C programming language should not have too much trouble.

- An Implicit Send DLL provides five functions:
 - `InitImplicitSend()`
 - `ImplicitSendStyle()`

- `O ImplicitSendPinCode()`
- `O ImplicitStartTestCase()`
- `O ImplicitTestCaseFinished()`

All five functions must be provided. If any one of the functions is missing, or has an incorrect name, PTS will be unable to load the DLL. The functions are described in the following section (3.2, "Implicit Send functions").

- Implicit Send DLLs are loaded dynamically. They are standalone entities that do not need special names or folder locations in order for PTS to locate them. (A configuration setting tells PTS where to find the DLL.)

Implicit Send functions

Conventions

Each of the function definitions contains the following two declarations. These declarations must be used in order for the interface between PTS and an Implicit Send DLL to work correctly.

- `extern "C"` – This declaration tells the C++ compiler that the symbol name for a function is not to be decorated in any way. The C++ language allows multiple functions with the same name, as long as they have different parameter lists and/or return types. This is accomplished by changing – or “decorating” – the function names behind the scenes, resulting in each function actually having a different name.

PTS expects the functions in an Implicit Send DLL to have plain, undecorated names.

- `WINAPI` – This declaration tells the C++ compiler that the function calling convention is the same as functions defined in the Windows API. This primarily has an impact on parameter handling during the calls from PTS to the Implicit Send functions.

InitImplicitSend()

Declaration: `extern "C" bool WINAPI InitImplicitSend(void);`

Parameters: None

Return values: "true" if successful
 "false" if not successful

This function is called during the initialization of an Executable Test Suite (ETS), just after the Implicit Send DLL has been loaded into memory.

It can be used to perform any initialization that might be needed before executing test cases.

If no initialization is required, the function can simply return a value of "true".

If the initializations failed, which would lead to the DLL not being usable, a "false" value should be returned. In this case, the ETS will be disabled.

ImplicitStartTestCase()

Declaration: extern "C" void WINAPI ImplicitStartTestCase(std::string& strTestCaseName);

Parameters: A character string containing the name of the current test case

Return values: None

ImplicitStartTestCase() is called at the start of each test case execution. It provides the name of the test case that is starting.

This function can be used to perform initializations that are needed at the start of every test case. The test case name allows the initialization process to be customized to specific test cases.

ImplicitSendStyle()

Declaration: extern "C" char* WINAPI ImplicitSendStyle(std::string& strMmiText, UINT mmiStyle);

Parameters: strMmiText – Information about the MMI (Implicit Send) request being made
 mmiStyle – A value describing the type of request and the expected values

Return values: A pointer to a character string containing the information to be returned
 A NULL pointer if the request cannot be processed or no information is to be returned

This is the main routine for handling Implicit Send requests; the majority of interactions with the test operator or automated test environment will be handled by this function.

strMmiText will consist of two pieces

- A message “tag” that uniquely identifies the message (see [Message tags](#)).
- Message text that would normally be displayed to the test operator.

In the default Implicit Send DLL used by PTS, the mmiStyle parameter is used to select the style of dialog box that is to be displayed. Custom DLLs can use this information to determine how to process the strMmiText, the type of request that is being made, and the expected return values.

In most cases, the return value will be a pointer to a string containing the word “OK”. Some requests, such as those using MMI_Style_Edit1 expect a string of data – for example, a PIN code or a file name – as the return value.

The MMI_Style_Edit2 style provides a list of items in the strMmiText and expects one of those items to be returned.

For more information, see [MMI styles](#).

Scope of the return value

It is important to note that the character string that is pointed at by the ImplicitSendStyle() return value must not go “out of scope”. For example, std::string values that are created during the execution of a function are likely to be destroyed when the function exits. The returned pointer may continue to point at valid text, but there is a good chance that the memory space used by the string could be reused.

One way to avoid this type of issue is to create the string to be returned in dynamic memory (using malloc() or “new”). The string would then be added to a list or variable sized array (such as a std::vector).

All of strings returned during the execution of the test case would remain until the end of the test, at which time they could be destroyed.

For an example of one way to do this, look in sample source code (see [Sample source code](#)) at the PersistentText C++ class (PersistentText.cpp/.h) and how the class is used in ImplicitSend.cpp. In this example, the most recently returned string is “persistent” and is deleted when the Implicit Send DLL is unloaded.

ImplicitSendPinCode()

Declaration: extern "C" char* WINAPI ImplicitSendPinCode(void);

Parameters: None

Return values: A pointer to a character string containing a PIN code to be returned
 A NULL pointer if the request cannot be processed or no PIN code is to be returned

This is a special case function that is only used when a dynamic PIN code is needed.

It is not currently used in PTS, but it might be used in the future. One way to implement this function in order to be prepared for future use is

```
extern "C" char* WINAPI ImplicitSendPinCode(void)
{
    std::string strPrompt = "Please enter a PIN Code:";
    return(ImplicitSendStyle(strPrompt,MMI_Style_Edit1));
}
```

Please refer to [Implicit Send functions](#) for details regarding the return value from this function.

ImplicitTestCaseFinished()

Declaration: extern "C" void WINAPI ImplicitTestCaseFinished(void);

Parameters: None

Return values: None

ImplicitTestCaseFinished() is called at the end of test case execution. It may be used to perform any cleanup that is needed, or to undo operations that were performed during ImplicitStartTestCase().

Final cleanup

The Implicit Send API does not provide a final cleanup function. Generally such a function is not needed because the unloading of the DLL or the termination of the main PTS executable causes resources such as open files to be closed and dynamic memory to be released.

Should some form of final cleanup be required, the following is suggested:

1. Create a C++ class that contains the various objects that need to be cleaned up when the DLL is unloaded.

2. Declare an instance of the class at module level scope and make sure that the class is visible to all functions that need to access the data within. Variables declared at “module level scope” are those that are declared somewhere in a source file, but outside the boundary of all of the functions in the file.
3. Place the necessary cleanup code in the class destructor. When a DLL (or executable program) is about to be unloaded from memory, the C++ runtime support invokes the destructor for each instance of a class that is defined at module level scope. Placing the cleanup code in the destructor ensures that it executes at the proper time.
4. Note that the standard C++ “singleton pattern” should not be used here. The standard singleton pattern uses a pointer to an instance of a class. The runtime support cleanup code will NOT call the destructor for an object that is accessed indirectly via a pointer.

As mentioned in [Implicit Send functions](#), the sample source code for the default Implicit Send DLL uses this mechanism to address the `ImplicitSendStyle()` return value “persistence” issue.

Another possibility is to use a `DllMain()` function and perform the cleanup work when it is called with a reason code of `DLL_PROCESS_DETACH`. Note however that the use of `DllMain()` is no longer recommended by Microsoft. When using current versions of the Microsoft development tools, `DllMain()` isn't even required – the language runtime support provides its own.

Message tags

As mentioned in [Implicit Send functions](#), the `strMmiText` parameter passed to `ImplicitSendStyle()` is a string consisting of two parts. One of those parts is a message tag that uniquely identifies the message.

The purpose of the message tags is that they will always be the same regardless of the informational text that may be displayed to a test operator. This means that custom Implicit Send DLLs do not need to process the informational text – they only need to process the tag in order to know what request is being made.

The message is at the beginning of the `strMmiText` string and is in the following format

```
{<message number>,<test case name>,<test suite name>}
```

where

- <message number> identifies a message within a given test suite;
- <test case name> identifies the executing test case;
- <test suite name> identifies the Executable Test Suite that contains the currently executing test case.

The combination of <message number> and <test suite name> uniquely identifies a message across all Executable Test Suites. For example

```
{999,<any test case name>,OPP}
{999,<any test case name>,FTP}
```

are different messages even though they have the same <message number>.

The <test case name> helps to identify the usage of the message. For example, in the Object Push Profile (OPP) test suite <message number> 47 is used in every test case where the test operator (or automated test platform) needs to confirm that an object transfer occurred successfully. In test case TC_SERVER_OPH_BV_03_I the operator needs to confirm that a new contact entry with the name "OPHBV03" is on the IUT. In TC_SERVER_OPH_BV_07_I, a new calendar entry titled "OPHBV07" should have been created.

An automated test platform can distinguish between the two uses of <message number> 47 by looking at the <test case name>

```
{47,TC_SERVER_OPH_BV_03_I,OPP}Please check that ...
```

```
{47,TC_SERVER_OPH_BV_07_I,OPP}Please check that ...
```

Finding the tags

The default Implicit Send DLL provided with PTS removes the tags before sending the message text to the popup dialog. In other words, in normal operation PTS does not display the message tags.

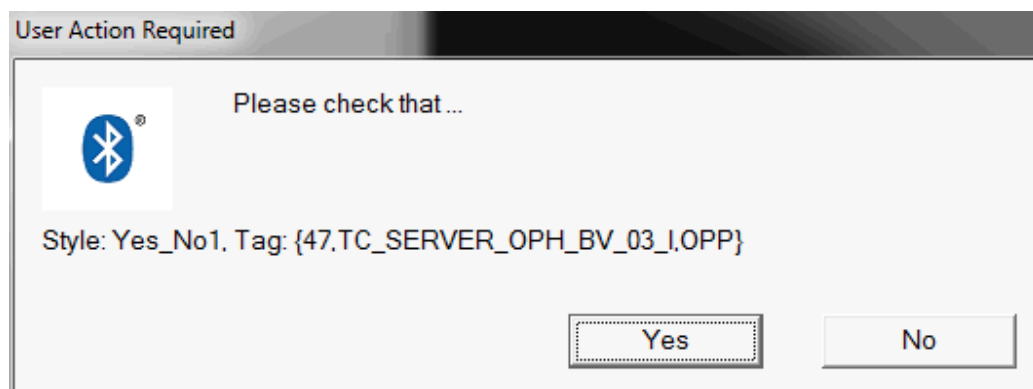
There are three ways to determine the tag associated with a given message:

1. Have the default Implicit Send DLL display the tag

Starting with version 4.5.3 of PTS, the default Implicit Send DLL has the option to display both the style of each message along with its tag. This functionality is enabled by adding the following lines to PTS.ini, normally found in C:\Program Files\Bluetooth SIG\Bluetooth PTS\bin.

```
[ImplicitSend]  
showTag=1
```

For the example above, this setting will cause the dialog to look something like this:



2. Consult implicit_send_log.txt

The default Implicit Send DLL creates a log containing all of the Implicit Send requests that have been made since the workspace was created. The file is named implicit_send_log.txt and may be found in the workspace folder.

For example, if the workspace for the test that was executed above is named "OPP Profile", the workspace folder might be

C:\Program Files\Bluetooth SIG\My Workspaces\OPP Profile

Looking in the implicit_send_log.txt file for the above message would find this entry

```
-----
Style: Yes_No1, Tag: {47,TC_SERVER_OPH_BV_03_I,OPP}
Please check that ...
```

3. Consult the ATS document

Each test suite has an accompanying Abstract Test Suite (ATS) document that describes the details of the suite and its environment. In each ATS document there is a section on Implicit Send that includes a table listing each of the messages including their tags.

For example:

TSC_MMI_Confirm_Preparation_Template	"{47,%s,OPP}Please check that <what to check for> This test case will <what test does> Press OK when you are ready to continue. Press CANCEL if you want to terminate this test case."
--------------------------------------	--

The "%s" in the message tag is replaced with the name of the currently executing test case at runtime.

MMI styles

The mmiStyle parameter to ImplicitSendStyle() provides direction about the contents of the strMmiText parameter along with an indication of the expected return value. When used with the default PTS Implicit Send DLL, the mmiStyle value selects the type of dialog box that will be displayed along with the buttons that will appear.

mmiStyle 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

“Simple prompt” message type

All of these MMI styles are used to instruct the test operator or automation test platform to take an action. The action may be to make a connection from the IUT to the PTS, press a button on the IUT, etc.

For these messages the strMmiText contains instructions about the action that is needed.

If the action can be successfully completed, the return value from ImplicitSendStyle() should be a pointer to a character string such as “OK”. The actual contents of the string don’t matter. Please be sure to take a look at [Implicit Send functions](#) for important information about the “scope” of the string that is returned.

Successful completion is indicated in the default Implicit Send DLL when the user presses the OK, Yes, Retry or Ignore buttons.

If the operation cannot be completed, or the proper response to an action is to indicate that it did not happen, ImplicitSendStyle() should return a NULL pointer.

“Request for data input” message type

This message style is used when information is needed from the test environment, and that information is not available until after the test case begins execution. For example, the Passkey Entry association model in Secure Simple Pairing requires that one device display a six digit number. The number must be entered on the other device to complete the association process. The number itself is random and is not generated until the Secure Simple Pairing process begins.

strMmiText describes the information that is being requested.

The return value from `ImplicitSendStyle()` should be a pointer to a character string containing the requested data, if the data is available. If the requested data is not available, or an error occurs, a NULL pointer should be returned.

As has been noted above, the string pointed at by the return value should not be allowed to go out of scope. ([Implicit Send functions](#).)

“Select item from a list” message type

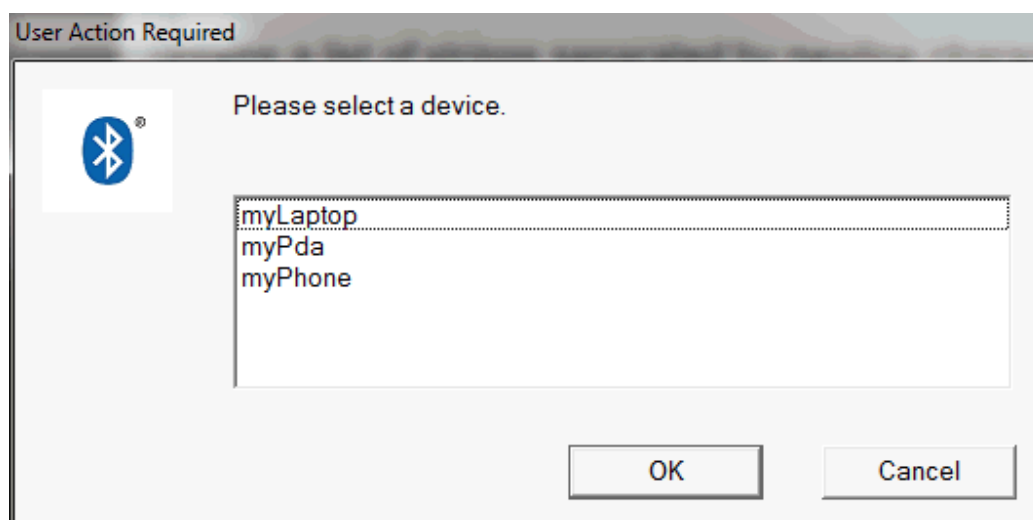
For this message strMmiText contains a list of strings separated by newline characters. (C/C++: `'\n'`, ASCII code 0x0A.) The first string in the list contains the instructions to the user. The rest of the strings provide a list values for selection. The list is ended by an empty line.

For example, strMmiText may contain the following information

```
"{<message tag>}Please select a device.\nmyPhone\nmyLaptop\nmyPda\n\n"
```

The first item in the list ("Please select a device.") indicates that a list of devices follows and that one of the devices should be selected. There are three devices in the list: "myPhone", "myLaptop", and "myPda".

The items in the list are separated by newline characters, indicated as `'\n'` above. The extra `'\n'` at the end of the string marks the end of the list.



The expected return value is a pointer to a string containing one of the values from the list. A NULL pointer should be returned in case of error or if none of the values are appropriate at that point in the test.

The string to be returned will need to be a copy of one of the items in the list. Returning a pointer to the first character of one of the items in the list will not work since there is additional information (new lines and other items) following the selection. The copy is subject to the data scoping concerns mentioned in [Implicit Send functions](#).

Software build requirements

It was mentioned earlier ([Basic information](#)) that Implicit Send DLLs must be written in C++. A few additional requirements need to be considered when starting the development of an Implicit Send DLL.

- Microsoft Visual C++ must be used. C++ objects such as `std::string` are not guaranteed to be implemented the same way in every compiler. Mixing definitions is a recipe for trouble.
- Microsoft Visual C++ 2008/Visual Studio 2008 must be used for development of a custom Implicit Send DLL. Subtle runtime issues can occur when mixing different versions of the Visual C++ runtime environment.

In particular, Visual C++ 2010 has been found to produce an Implicit Send DLL that does not work with PTS.

- The PTS Team suggests that custom DLLs be built in the "Release" configuration. Data structures and dynamically allocated memory may be laid out differently between "Debug" and "Release" configurations. (The "Debug" versions may contain extra elements to assist in the debugging process.) It is rarely a good idea to mix "Release" and "Debug".

Note that it is possible to use the Visual Studio Debugger on executables and DLLs that are built in the "Release" configuration. A few small changes may be needed to the Visual Studio project configuration to enable this functionality. Please contact PTS Technical Support for more information.

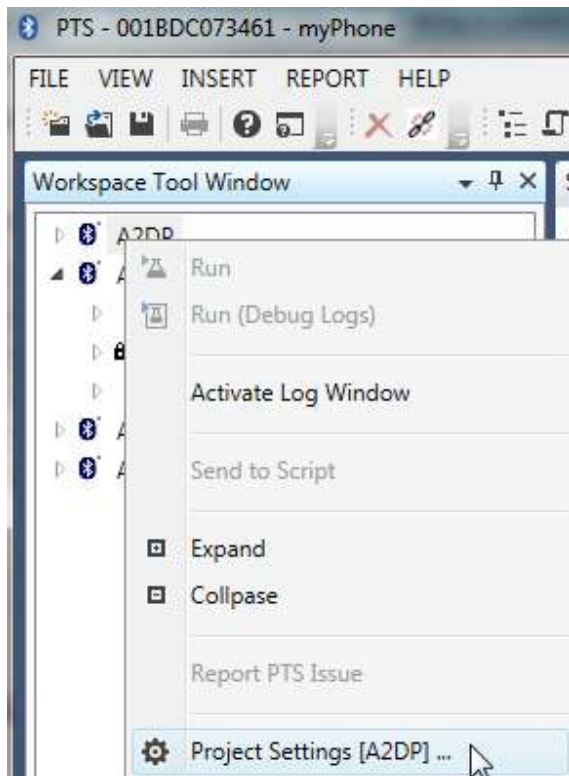
The requirements above are a result of the development environment used by the PTS Team. The Team uses Microsoft Visual C++ 2008 and the PTS executables and DLLs are built in the "Release" configuration.

Activating a Custom Implicit Send DLL

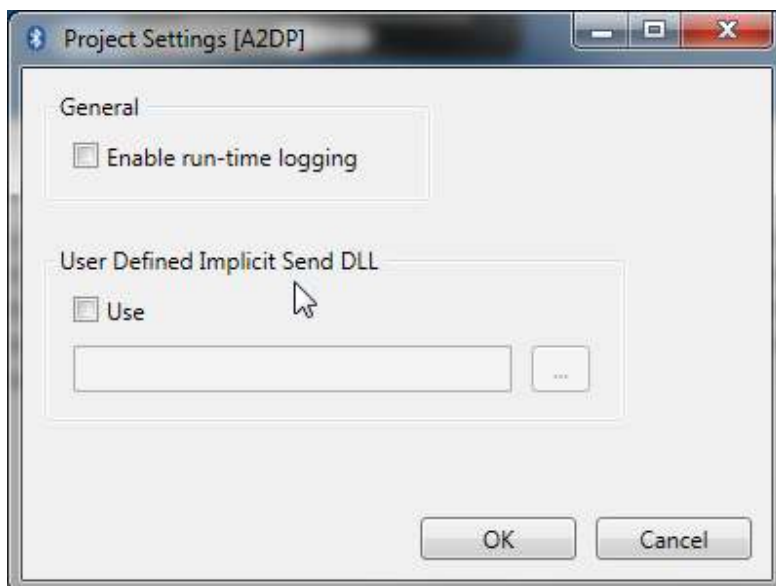
Activating a custom Implicit Send DLL

Once a custom Implicit Send DLL has been created, it is a fairly simple matter to start using it with PTS. The DLL may be attached to a test suite selected by a PTS project via the Project Settings dialog.

Begin the process by selecting the desired project (test suite) in the Workspace Tool Window. Right click on the top level node of the project and select "Settings" from the menu that appears.



At the bottom of the Project Settings menu is a section labeled "User Defined Implicit Send DLL". Normally the box labeled "Use" is unchecked. When this box is unchecked the test suite will use the default Implicit Send DLL provided by PTS.



Place a check mark in the "Use" box. This will cause the text box and browse button to become active. Enter the path to the custom Implicit Send DLL in the text box or browse button ("...") to locate the DLL. After the DLL has been selected, press the OK button to record the change.



The custom Implicit Send DLL will be used starting with the next test case that is executed in the project.

Usage Notes

- The custom Implicit Send DLL may be disabled at any time by returning to the Project Settings page and removing the checkmark from the box labeled "Use".
- The procedure above attaches a custom DLL to one and only project in a single workspace. The process must be repeated to use the DLL with other projects in the same workspace, or, with the same test suite (project) in a different workspace.

Technical Tidbits

Automatic dismissal of Implicit Send requests

At various points during the execution of a test, the test case implementation can detect that an action requested via Implicit Send has occurred. When this happens, the test case may attempt to complete the `ImplicitSendStyle()` or `ImplicitSendPinCode()` operation that is in progress.

This most commonly occurs with messages using `MMI_Style_Ok_Cancel2`. These messages tend to be "transient", for example, "Using the IUT, make a connection to the PTS". For these types of actions, there is no need to require operator interaction with PTS. The operator can simply take whatever steps are necessary on the IUT to cause the connection to happen; the test case can then detect the connection and take down the dialog.

The mechanism used to do this is to send simulated button presses to a dialog whose title is "User Action Required". This works very well with the default Implicit Send DLL because all of the dialogs it displays are titled "User Action Required".

This however may not work very well with custom Implicit Send DLLs – especially ones that have no need to create popup dialogs. The test case will send the simulated button presses, but no one – most specifically the functions in the custom DLL – will be listening.

This situation may not be as bad as it seems. In all likelihood the IUT also knows that the requested action has taken place and would normally notify the user of the device. When the user of the device is replaced with an automation test platform, the platform can detect the situation and notify the custom Implicit Send DLL.

If it turns out that a custom DLL needs to know when the simulated button presses occur, it could create a hidden window whose title is "User Action Requested". This would allow the delivery of the simulated button presses to the custom Implicit Send DLL.

For more information about the simulated button presses, please contact PTS technical support.

ImplicitSend() function

Earlier versions of PTS used a function called ImplicitSend(). This function has been replaced by ImplicitSendStyle() and is no longer used by the PTS.

TSPX_use_implicit_send

Most test suites have a IXIT value named TSPX_use_implicit_send which is used to enable or disable the Implicit Send functionality. Normally the value of this item is TRUE indicating that Implicit Send is to be used.

Setting the value to FALSE will disable Implicit Send for both user developed DLLs and the PTS default DLL.

The value of TSPX_use_implicit_send should be checked whenever it appears that the Implicit Send functionality is not working at all.

Sample Source Code

The PTS installation has a folder containing source code that may be used as a reference during development of a custom Implicit Send DLL. The source code itself is the complete source for the default Implicit Send DLL that is normally used by PTS. A Visual Studio project is included making it possible to build and execute the sample.

Of particular interest is implicit_send.cpp. In addition to the functions it provides, there are notes for an alternate implementation that connects to an automated test platform via TCP/IP.

The sample may be found under custom\implicit_send in the PTS installation folder. The default path to this location is

C:\Program Files\Bluetooth SIG\Bluetooth PTS\Custom\implicit_send

One DLL or many DLLs?

The tag data in the Implicit Send messages makes it possible to use the same DLL for more than one profile since each message is uniquely identified by <message number>, <currently executing test case>, and <currently active test suite>. ([Message Tags](#))

It may be convenient to develop one Implicit Send DLL for all uses and use the test suite name in the message tag to know which profile is requesting assistance. On the other hand, a single DLL may be more complicated to construct and maintain, suggesting that a custom DLL for each test suite might be more appropriate.

The point to keep in mind is that PTS can support either design decision – developers of custom Implicit Send DLLs are not locked into one way or the other.

Hybrid environments

It may be desirable to replace some, but not all, of PTS's default Implicit Send handling. This is possible by doing the following:

- Create a custom Implicit Send DLL.
- In the custom DLL, InitImplicitSend() could dynamically load the default DLL using the Windows LoadLibrary() and GetProcAddress() API functions.
- When a message arrives at ImplicitSendStyle() in the custom DLL, the function could look at the message tag and decide whether or not it wants to handle the message. If the custom ImplicitSendStyle() does not want to handle the message, it could call ImplicitSendStyle() in the default DLL using the same parameters.

The return value from the default DLL would then become the return value for the custom DLL.

Extended Automating

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.
2. This feature can be used with either of the following program control features and is described in the "Automating – Using Implicit Send" reference document.
3. "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.
4. The "Scripting" reference document describes this feature.
5. "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".

"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\Bluetooth SIG\Bluetooth PTS\Custom\PTSControlClient

folder in the PTS installation.

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 the User Interface will appear. Users should pretend that it is not there as interactions with the User Interface can interfere with the PTS Control API.

There are two ways to start PTS in COM Server mode:

1. Do nothing.

After the Component Object Model has been initialized (via `CoInitialize()` or `CoInitializeEx()`) the Client Application needs to 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.)

If PTS is currently running and is in COM Server Mode, the call to `CoCreateInstance()` will simply connect to the running instance.

2. Start PTS manually.

PTS can be started in COM Server mode from a command prompt by adding the flag “-autotest” to the end of the command line.

```
"C:\Program Files\Bluetooth SIG\Bluetooth PTS\bin\PTS.exe" -autotest
```

General 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.)

Functions in the PTS Control API

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.

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.

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>.pqw" in the "root" of the Workspace folder.

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.

Working with Projects

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

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 [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.

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 [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);
```

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.

A value greater than or equal to zero if successful.

Return values A value less than zero if not successful.

For a list of error codes specific to the PTS Control API see [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.

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.

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 [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.

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 [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";  
  
LPWSTR pszTestCaseName;  
  
ppszTestCaseName = 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);
```

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 [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":
LPWSTR pszTestCaseDesc;
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);
```

IsActiveTestCase()

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

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

pblsActive: 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 [API Error Codes](#).

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

RunTestCase()

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

Declarations: 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 [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 [Logging and unattended operation](#) for more information.

StopTestCase()

Declaration: HRESULT StopTestCase();

Parameters: None.

Return value: PTSCONTROL_E_FUNCTION_NOT_IMPLEMENTED

StopTestCase() is not currently implemented and calls to it will always return the status code above.

Working with ICS and IXIT data

UpdateICS()

Declaration: HRESULT UpdateICS(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 [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.

UpdateIXITParam()

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: 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 [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)

Logging and unattended operation

During normal operation PTS sends informational output 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.

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.

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_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 PTSTestControl.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 PTSTestControlClient 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. 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.
PTS_LOGTYPE_PRELIMINARY_VERDICT	 "Preliminary Verdict" messages can be used to determine at what point during execution that a Test Case failed. 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	 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.

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 [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.

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

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

IPTSImplicitSendCallbackEx object

The IPTSImplicitSendCallbackEx 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 IPTSImplicitSendCallbackEx derived object needs to be registered with the PTS Control API via the RegisterImplicitSendCallbackEx() function.

IPTSImplicitSendCallbackEx::OnImplicitSend()

HRESULT OnImplicitSend(LPCWSTR pszProjectName, WORD wID,

Declaration: LPCWSTR pszTestCase, LPCWSTR pszDescription, DWORD style,

LPCWSTR 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 that 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 pszResponse 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

[MMI Styles](#) 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 `wscpy_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;
```

RegisterImplicitSendCallbackEx()

Declaration: HRESULT RegisterImplicitSendCallbackEx(IPTSImplicitSendCallbackEx* pCallback)

Parameters: pCallback: A pointer to an instance of an IPTSImplicitSendCallbackEx 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 [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.

UnregisterImplicitSendCallbackEx()

Declaration: HRESULT UnregisterImplicitSendCallbackEx(IPTSImplicitSendCallbackEx* pCallback);

Parameters: pCallback: A pointer to an instance of an IPTSImplicitSendCallbackEx 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 [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.

IPTSImplicitSendCallback object

This object is provided for backwards compatibility with existing Client Applications and use of the IPTSImplicitSendCallbackEx 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.

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 IPTSImplicitSendCallbackEx::OnImplicitSend() for more information about these parameters.

Limitations with IPTSImplicitSendCallback::OnSend()

IPTSImplicitSendCallback::OnSend() has a few limitations that have been addressed by IPTSImplicitSendCallbackEx::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.

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 [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.

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 [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.

General information functions

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 [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.

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 [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.

Sample Program - PTSControlClient

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\Bluetooth SIG\Bluetooth PTS\Custom\PTSControlClient

and the executable can be found at

C:\Program Files\Bluetooth SIG\Bluetooth PTS\bin\PTSControlClient.exe

Preparing to use PTSControlClient

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

1. Creating a Workspace

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

To create the Workspace run PTS in the normal way and create a new workspace as usual. Or, if a suitable Workspace already exists it can be used.

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

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 PTS Control Client 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\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\Bluetooth SIG\Bluetooth PTS\Documentation\Automation\TestScriptTemplate.xlsx

and follow the instructions.

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>
```

Running the Test Script

Once the Test Script is ready, it can be executed by running PTSTestControlClient, 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 PTSTestControlClient application.

"C:\Program Files\Bluetooth SIG\Bluetooth PTS\bin\PTSTestControlClient.exe" script.xml

API Error Codes

PTSTESTCONTROL_E_GUI_UPDATE_FAILED (0x849C0001)

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

PTSTESTCONTROL_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.

PTSCONTROL_E_FAILED_TO_CREATE_WORKSPACE (0x849C0003)

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

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.

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.

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.

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.

PTSCONTROL_E_TESTCASE_NOT_STARTED (0x849C0012)

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

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.

PTSCONTROL_E_PTS_VERSION_NOT_FOUND (0x849C0014)

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

PTSCONTROL_E_PROJECT_VERSION_NOT_FOUND (0x849C0015)

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

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.

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.)

PTSCONTROL_E_IXIT_PARAM_NOT_CHANGED (0x849C0021)

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

PTSCONTROL_E_IXIT_PARAM_UPDATE_FAILED (0x849C0022)

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

PTSCONTROL_E_TEST_SUITE_PARAM_UPDATE_FAILED (0x849C0022)

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

PTSCONTROL_E_IXIT_PARAM_NOT_FOUND (0x849C0023)

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

PTSCONTROL_E_ICS_ENTRY_UPDATE_FAILED (0x849C0030)

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

PTSCONTROL_E_ICS_ENTRY_NOT_FOUND (0x849C0031)

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

PTSCONTROL_E_ICS_ENTRY_NOT_CHANGED (0x849C0032)

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

PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_NOT_REGISTERED (0x849C0040)

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

PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_ALREADY_REGISTERED (0x849C0041)

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

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.

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().

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.

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.

Other error codes

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\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.

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.

Report Generator

Introduction

The Profile Tuning Suite (PTS) can be used throughout the lifecycle of a Bluetooth enabled product. If a Bluetooth product makes use of application profiles, then PTS must be used during the qualification of the product.

Many Bluetooth SIG members also use the PTS during initial software development and subsequent updates to a product. Use of the PTS during development can ensure that the Bluetooth functionality is operating correctly as features are added. The PTS is also useful in establishing baseline functionality that may be referred to later during regression testing when the product software is modified or enhanced.

A series of tests for a device will produce a lot of information. After the tests are complete, the question becomes "What to do with all of this data?"

PTS provides a report generator that can be used to consolidate the most useful information into a concise report.

Qualification test evidence

*The qualification process for a Bluetooth device consists of a number of steps. Some of these steps are

- Using the Test Plan Generator (TPG) and the Qualification Listing Interface (QLI), describe the Bluetooth features and functions that will be used by the device.
- Using the TPG, create a test plan that will be used to direct the testing of the device in order to demonstrate that it properly supports each Bluetooth feature and function that is used.
- Execute the test plan to collect proof that the device complies with the Bluetooth specifications, and report the results to the Bluetooth SIG using the QLI.

The reports produced by the PTS are in a format (XML) that can be processed by the QLI. The report generator in the PTS knows what information must be included when reporting test results and what information can be left out. This makes the use of the report generator quite convenient when it comes time to report the results of testing on a device.

In fact, because of this, results of qualification testing using the PTS must be submitted via a report created by the PTS report generator.

Development checkpoints

There are various points in the development of a product where it is useful to exercise the functionality. Use of the PTS at these times can ensure that the software in the device is tested the same way every time. In addition, the comparison of a set of test results with a previous set of test results can highlight problems that may have been introduced as development progressed.

Once again, the fact that the reports produced by the PTS report generator are in a computer readable format can be very helpful. The XML formatted information in the report can be imported into a database for convenient manipulation, analysis, and comparison.

Contents of a report

Each report produced by the PTS report generator contains test results for one project ("profile"). Multiple reports will need to be created when a device supports more than one profile, as indicated by the presence of multiple projects in the PTS workspace for the device.

Each report begins with the following information:

- The location where testing was performed and information about the person who performed the testing (referred to as "user information");

- A description of the device that was tested (referred to as a "device description");
- An overview of the test environment.

(This last item is often required in reports to regulatory agencies or compliance management organizations.)

The next part of the report contains:

- A summary of each of the test cases that were executed;
- A listing of the ICS and IXIT settings that were used while the device was being tested.

The remainder of the report consists of test results. This may be in one or two sections:

- For each test execution included the report, the first section of test case results lists
 - The test case name;
 - When the test was performed;
 - The result of the test execution ("verdict");
 - The version number of the test suite executable that was used;
 - A summary of the test execution;
 - Data referred to as the "Encrypted Verdict".

The "Encrypted Verdict" is a summation of the above information that has been encrypted for reporting purposes. Since this information cannot be changed, it is the primary evidence that a test case execution concluded successfully.
- A second section of test results may be present. This section contains the detailed test execution log for each of the test cases being reported on.

The inclusion the detailed logs is optional and is selected during the process of generating the report.

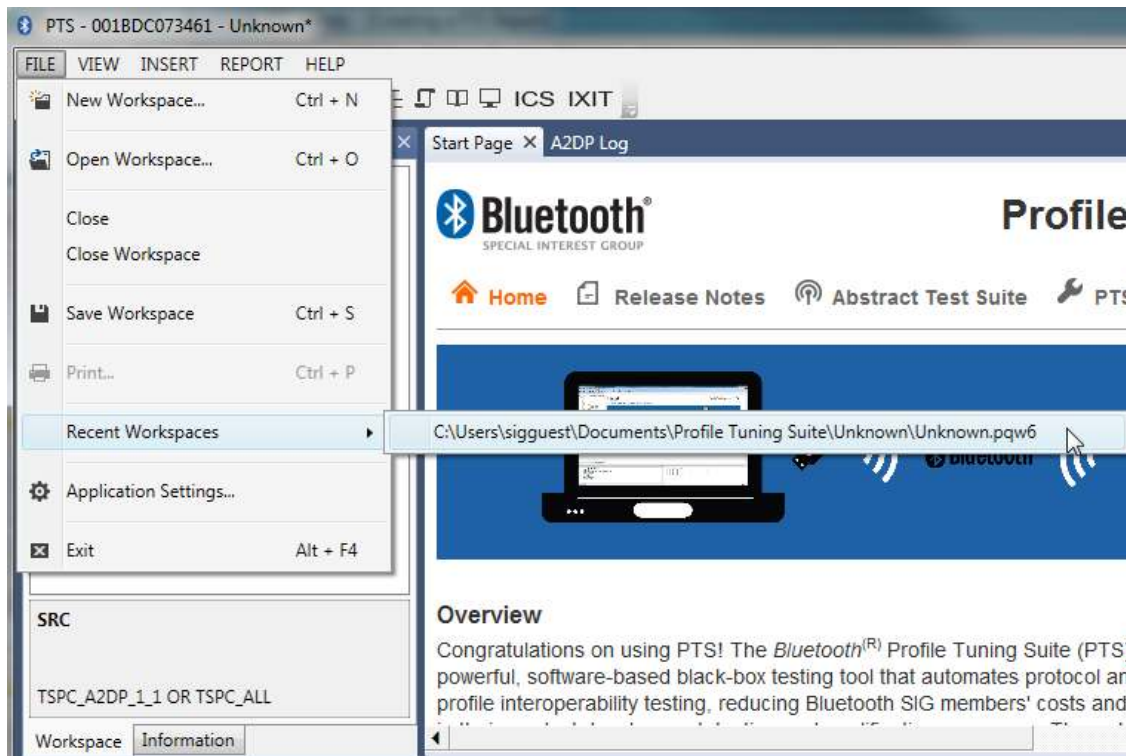
Creating a PTS Report

Once a series of tests have been run for a device, the process of creating a report is straightforward. The following steps are used to create a test report.

1. Select a workspace

Select the PTS workspace that contains the test results for the device. This is done in the normal way using the "Open Workspace..." or "Recent Workspaces" items on the "File" menu.

It is not necessary to select a project after selecting the workspace. The project to be used for the report will be selected from the "Report" menu.



2. Start the report generator

In the Report menu, click Generate or press Ctrl + R.



3. First time use of the report generator

PTS will store information about the last person who created a report using the report generator. In addition, the descriptions of devices for which reports have been created will also be stored.

When PTS is first installed on a new computer none of this information is present. The first time that the report generator is used after a fresh installation of PTS, the software will ask for information about the person creating the report and the description of the first device. This information can be edited later using features of the report generator main dialog.

Entering the initial user information

When using the report generator for the first time, fill out the information in the Report tab as completely as possible.

Report Generator

Report | **Products** | Test Cases

Product

Sig Product ▼

Add New

Tester Information

Company : Bluetooth SIG, Inc.

User Name : SIG Yser

Address :

Phone :

Email :

Options

Include Text Logging : ☐

Upload Report : ☒

QDID :

Generate Cancel

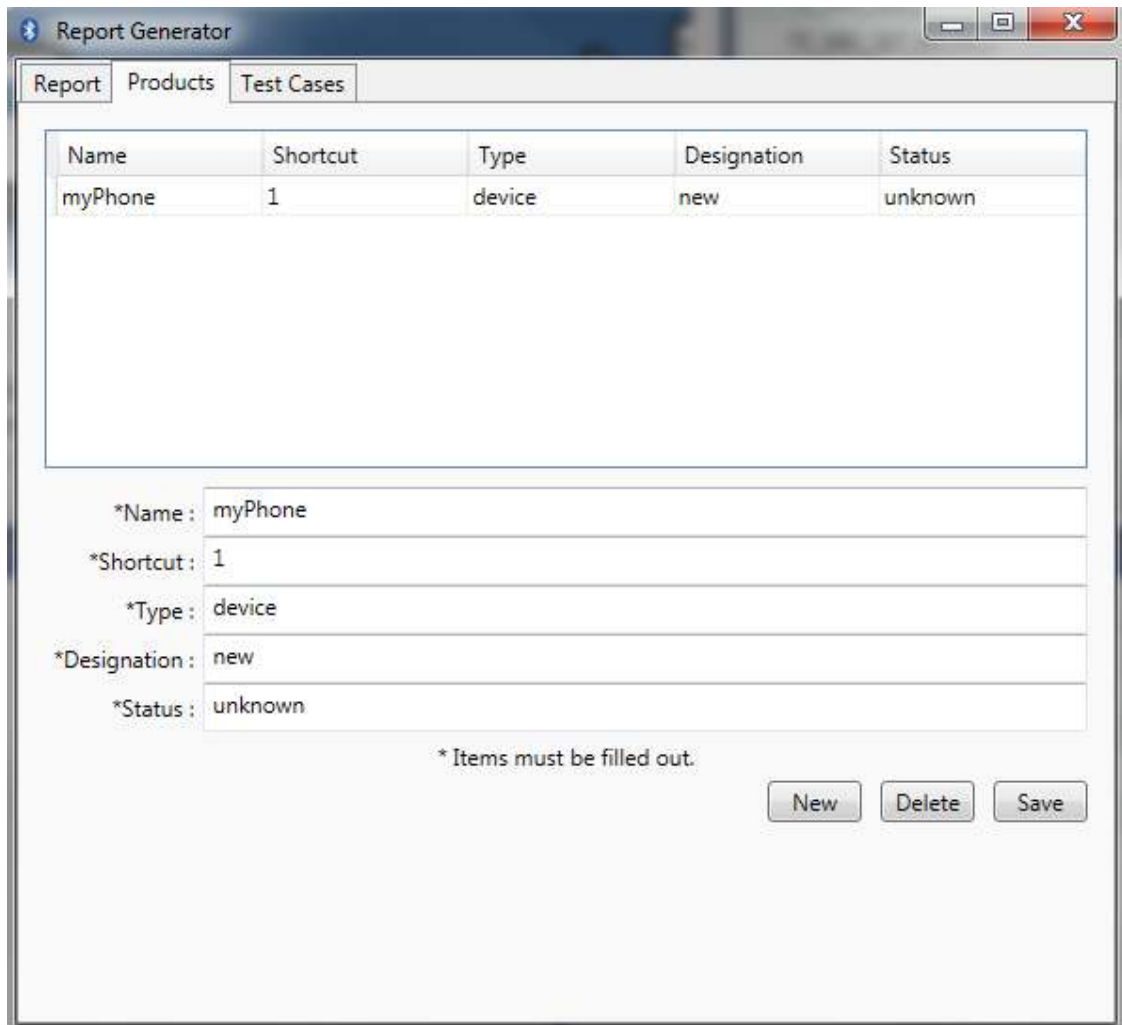
After entering the information, you can select a product from the Product drop down menu, or select the Products tab to enter information about the product.

Entering the description of the first device

The report generator needs at least one device description in order to generate a report. After entering the initial user information as described in the previous section, click the Products tab.

Enter the "Name" of the device and any other information as needed. When finished, click the "Save" button.

Only the "Name" item is required. The other items can be used to distinguish the device being described from similar devices that have been tested. All of the information entered in the "Product Details" dialog for a given device will appear in the report.



Name	Shortcut	Type	Designation	Status
myPhone	1	device	new	unknown

*Name : myPhone

*Shortcut : 1

*Type : device

*Designation : new

*Status : unknown

* Items must be filled out.

New Delete Save

It may be convenient to set the "Name" or the "Shortcut" to the name of the PTS workspace. This can make it easier to determine which workspace goes with which report.

After entering the description of the device, click the "Save" button. This will save the information and start the main dialog of the report generator.

4. Select the device description

The "Product" section at the top of the report generator dialog contains a dropdown list of showing the names of the stored device descriptions. The small downward facing arrow at the right of the list may be used to display that list.

Click on the device description of interest in order to select it.

If the description of the current device has not yet been entered, it may be created by clicking the "Add Product" button. (See [Adding and deleting device descriptions](#))

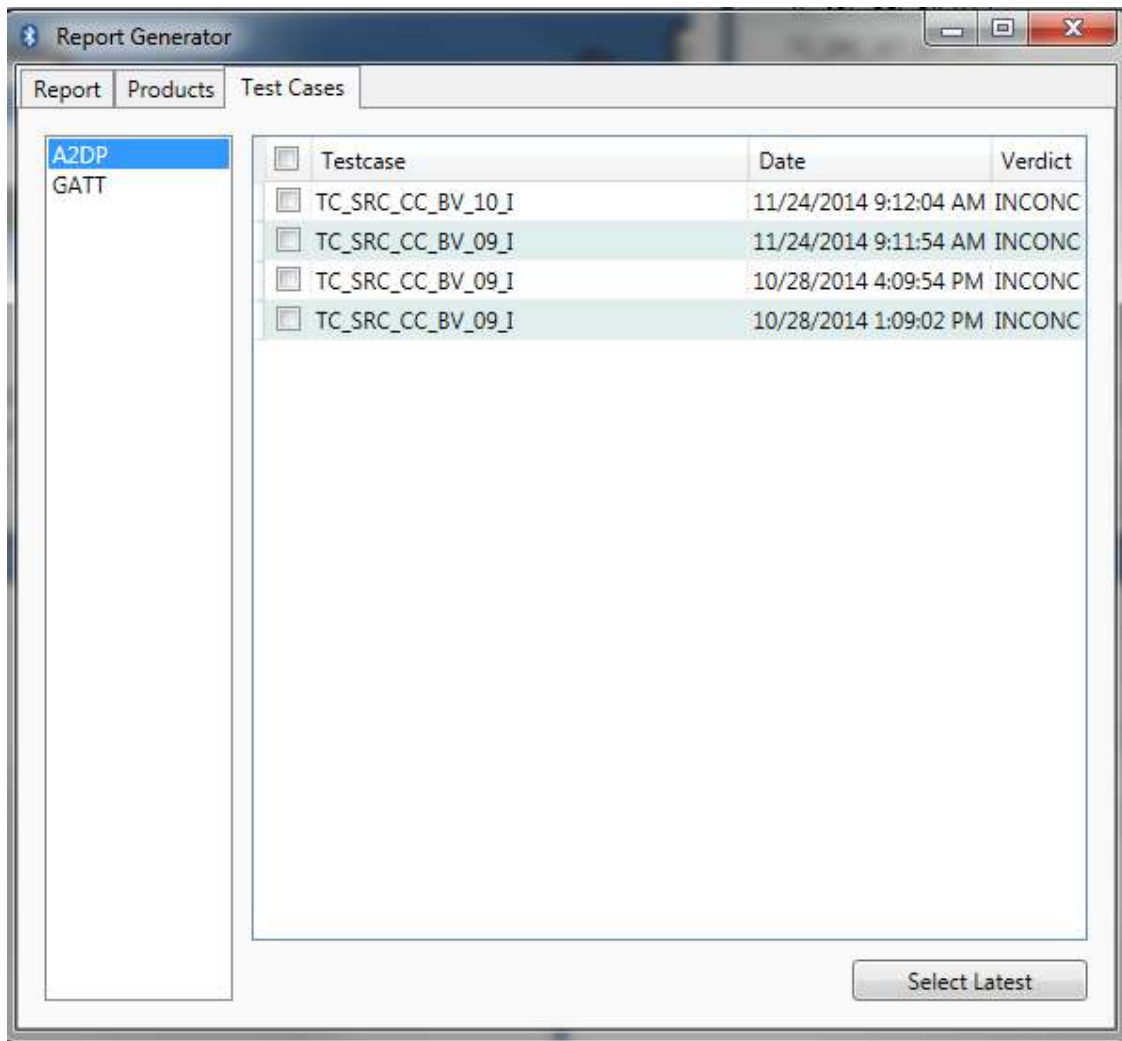
The screenshot shows a window titled "Report Generator" with three tabs: "Report", "Products", and "Test Cases". The "Report" tab is active. Inside the window, there is a "Product" section with a dropdown menu. The dropdown is open, showing two options: "myPhone" (which is highlighted in blue) and "Toaster". Below the product dropdown, there are several text input fields: "Company" (filled with "SIG Training Fake Company"), "User Name" (filled with "test user"), "Address" (filled with "123 Anywhere St Anytown, USA 98000"), "Phone" (filled with "000-000-0000"), and "Email" (filled with "john_doe@email.com"). Below these fields is an "Options" section containing two checkboxes: "Include Text Logging" (unchecked) and "Upload Report" (checked). There is also a "QDID" label followed by an empty text box. At the bottom right of the window are two buttons: "Generate" and "Cancel".

5. Selecting the test case results to be used in the report

A series of tests is likely to contain many executions of the same test case. A test case may have been executed more than once in order to exercise different settings of the ICS or IXIT data, or to retest various test cases after an updated software build. The next step in the report generation process is to select the test case results that should be included in the report.

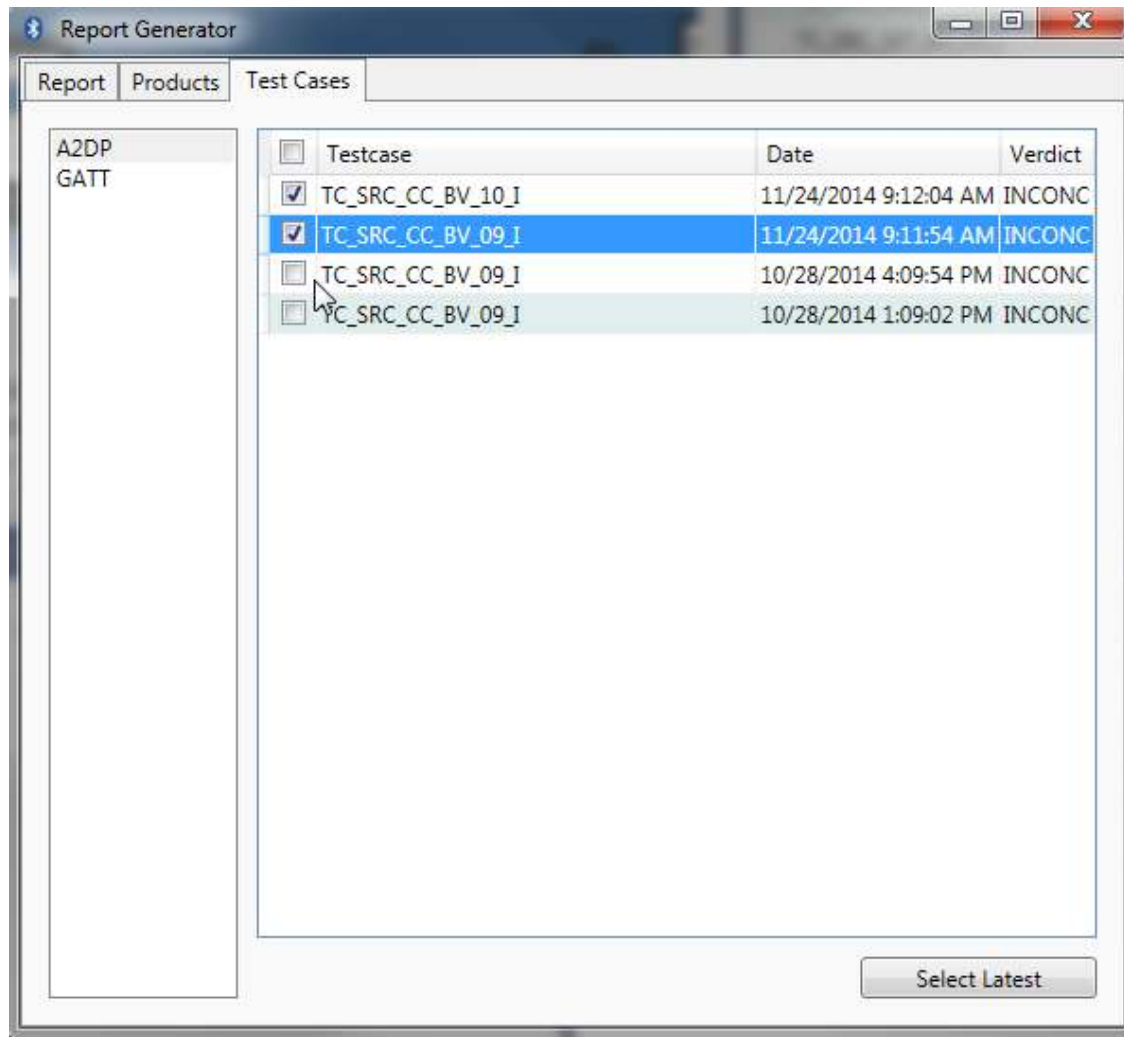
The "Select Test Cases" button will open a display listing all of the test case runs for which result data is available. The list may not include all of the test case executions that have been run for the current project -- some test results may have been deleted using the test case history editor. (See [Reviewing and Editing the Test History](#).)

The list of test cases will be in the order in which they were executed. When first opening the display, none of the test case results will be selected.



To choose the results to be included in the report, place a checkmark next to the entries that are of interest. You can either click on the checkbox or the item description.

The checkbox next to Testcase will select all the test cases. Unchecking the checkbox will clear the selections.



The "Select Latest" button may be used to select the most recent results for each test case. In many cases this will be the desired set of results.

Once the set of test cases to be included in the report have been selected, click the Generate in the Report tab.

Report Generator

Report Products Test Cases

Product

myPhone

Tester Information

Company : SIG Training Fake Company

User Name : test user

Address : 123 Anywhere St Anytown, USA 98000

Phone : 000-000-0000

Email : john_doe@email.com

Options

Include Text Logging : ☐

Upload Report : ☒

QDID :

Generate Cancel

It should be noted that the set of selected test cases is not "sticky". The set of test cases to be included in the report must be selected on every use of the report generator.

Which results to choose?

Generally, the most recent successful execution of each test case (ones with a "PASS" verdict) should be chosen for the report. These results indicate that the device was able to successfully execute the tests with the most recent ICS and IXIT settings.

On some occasions it may be desirable to include unsuccessful results in the report. Use of the Profile Tuning Suite is mandated for many of the profile level test cases. An issue in a test specification, or a software problem in the PTS may make it impossible for a given test case to result in a "PASS" verdict. When this happens, it needs to be shown that execution of the test case using the PTS was attempted. Inclusion of the unsuccessful result in the report will call attention to the failed attempt.

The inclusion of such results in the report can be used to help indicate that all of tests listed in the test plan created by the Test Plan Generator were attempted.

8. Including test execution logs in the report

The report created by the report generator may optionally include the detailed test execution log for each of the selected test cases. Inclusion of this information is not required for creating a report to be submitted as testing evidence via the Qualification Listing Interface.

Options

Include Text Logging : ☒
Upload Report : ☒
QDID :

If a report is to be used for internal documentation, or in the case that a test execution cannot result in a "PASS" verdict, it may be helpful to include the detailed logs in the report. Placing a checkmark in the box labeled "Generate Report with Text logging" will cause the test execution log for each of the selected test cases to be included in the report. Placing a checkmark in the box labeled Upload Report requires a valid QDID in order for the report to be uploaded.

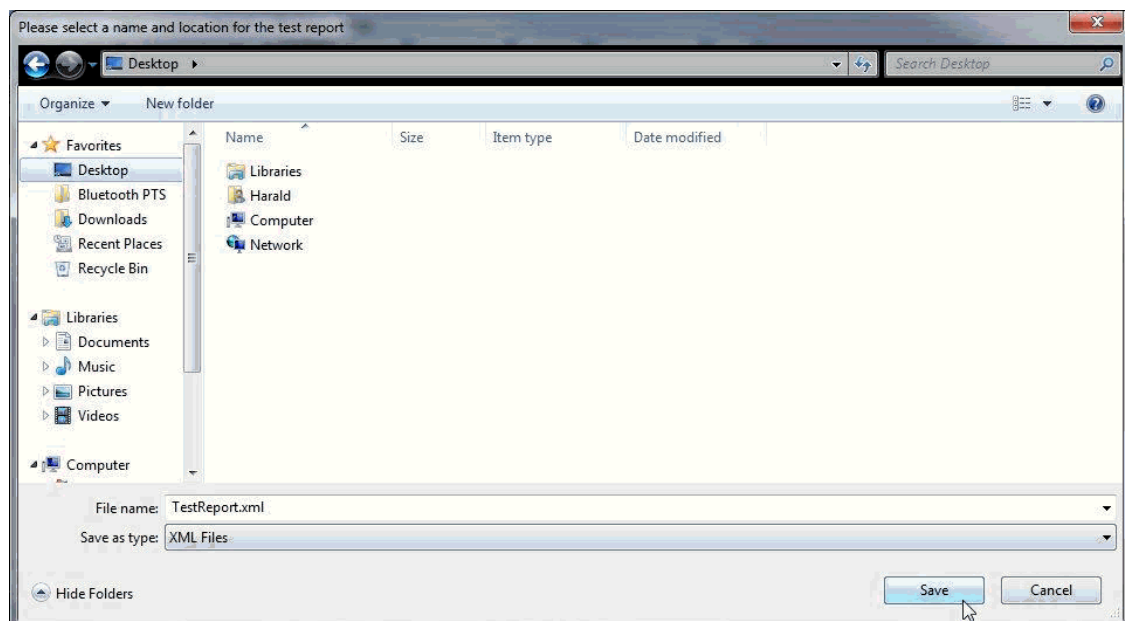
9. Generating the report

Once the test cases to be included in the report have been selected, and the user information and device description have been verified, it is time to create the report. The "Generate" button is used to start the creation of the report.

After clicking the "Generate" button, a dialog will appear asking where the file containing the report should be saved. Browse to the location where the report file should be stored.

The default name of the report file is "TestReport.xml", but this can be changed as needed. The default file extension of ".XML" should be left as is.

Once the location for the file has been selected and the file name has been specified, click the "Save" button to start the creation of the report.



As the report is being created, a progress bar will show the current status of the process.

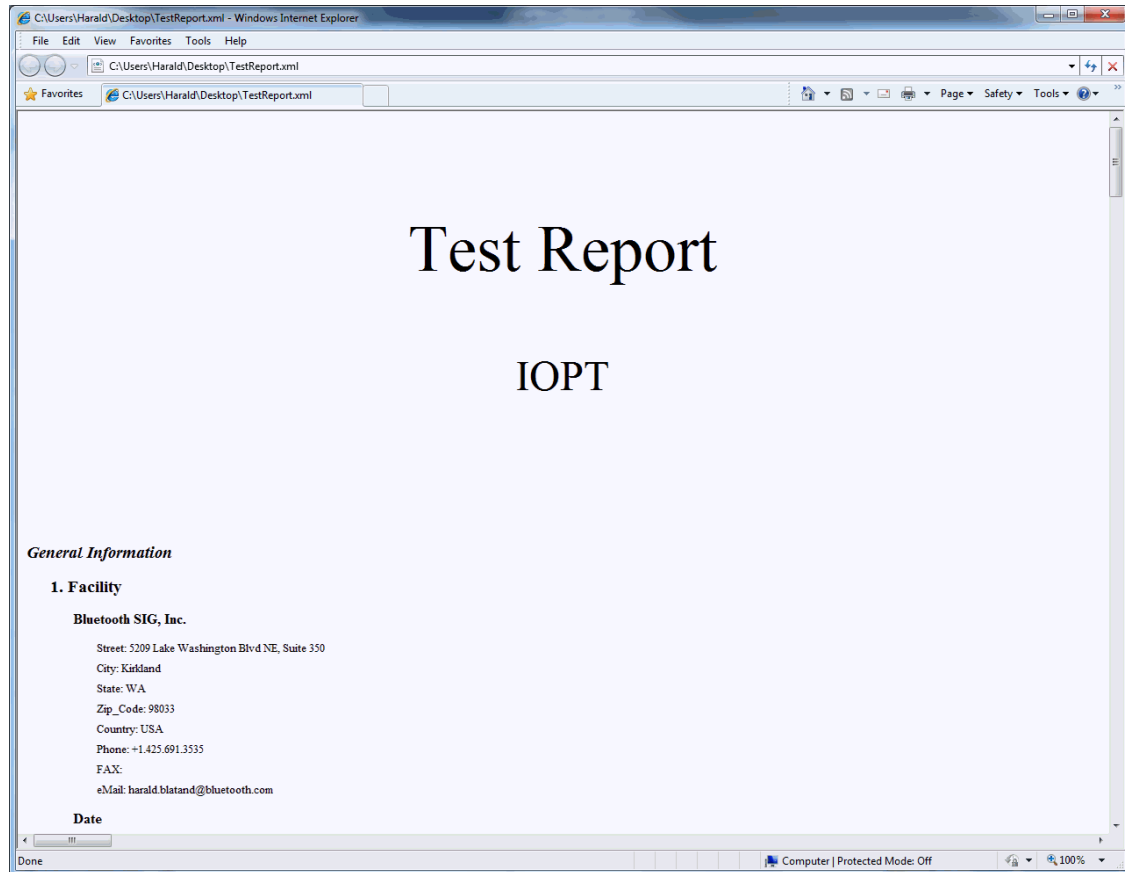
After the report is finished, the report generator will ask if the report should be viewed. Clicking "Yes" button will cause PTS to launch the application currently associated with the ".XML" file type. On many Windows® systems this application will be Internet Explorer®.

Report generation with text logging

The "Generating Detailed Log" phase can take quite a long time to complete when "Generate report with text logging" is selected. It may appear that the report generator has stopped running during this phase.

The number of test cases that have been selected for the report, and the size of the test execution logs for each of those test cases has a direct impact on the time it takes "Generating Detailed Log" to complete.

It may be the case that the report generator should be run overnight depending on the above criteria. A future release of PTS will provide better status information during this phase of the



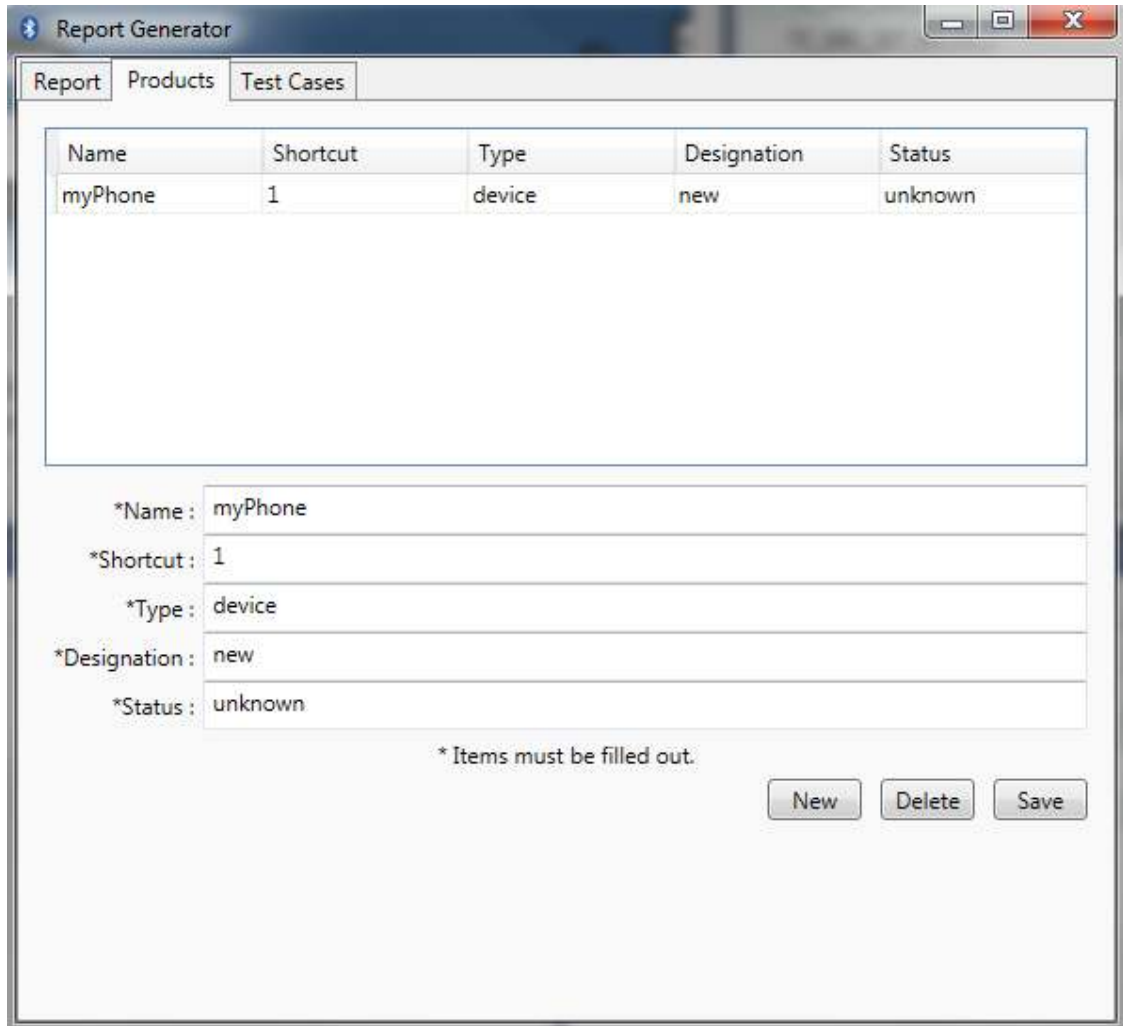
Adding and Deleting Device Descriptions

As mentioned earlier ([Creating a PTS Report](#)), PTS can store descriptions for more than one device. The list of stored descriptions is managed using two additional buttons in the Products tab of the Report Generator menu.

Adding a device description

Clicking the "New" button will clear the text fields and allow you to enter a new device description

After entering the details for the new device, click the "Save" button to store the information, or the "New" button to cancel the operation.



The screenshot shows a window titled "Report Generator" with three tabs: "Report", "Products", and "Test Cases". The "Report" tab is active. It contains a table with the following data:

Name	Shortcut	Type	Designation	Status
myPhone	1	device	new	unknown

Below the table, there are five form fields with labels and values:

- *Name : myPhone
- *Shortcut : 1
- *Type : device
- *Designation : new
- *Status : unknown

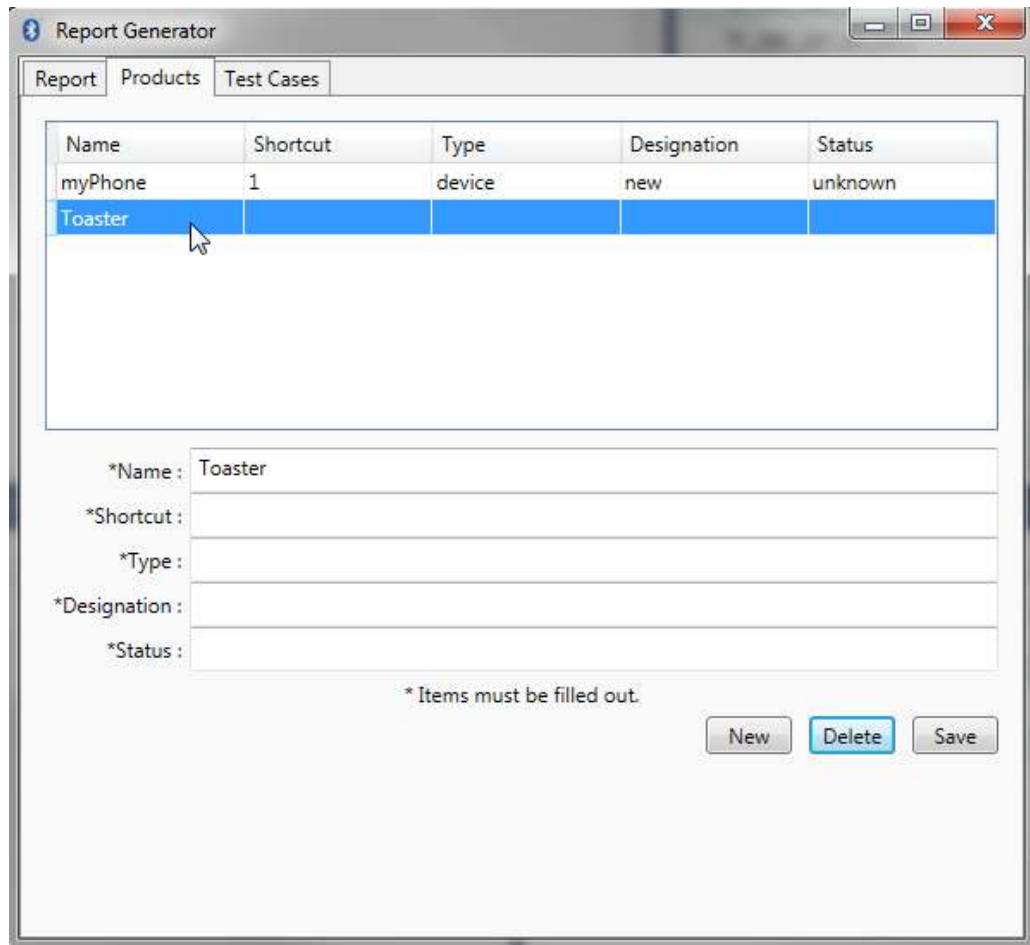
At the bottom right, there are three buttons: "New", "Delete", and "Save". A note at the bottom center states: "* Items must be filled out."

Note that the report generator will allow the creation of more than one device description with the same "Name". Doing this may be useful, but it does require that care be used when selecting the right device description for creating the report.

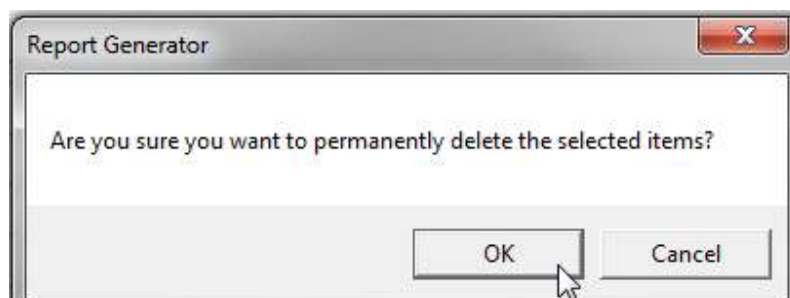
Deleting a device description

Any of the device descriptions may be removed from the stored list. There are four steps to this process:

1. In the Products tab, select the device description that is to be removed.



2. It is probably a good idea to review the description to make sure that it is the one that needs to be removed. This is especially true if more than one device description uses the same "Name".
3. Next, click the "Delete" button.
4. Finally, confirm that the selected description should be deleted by clicking "OK". Click "Cancel" to keep the selected device description.



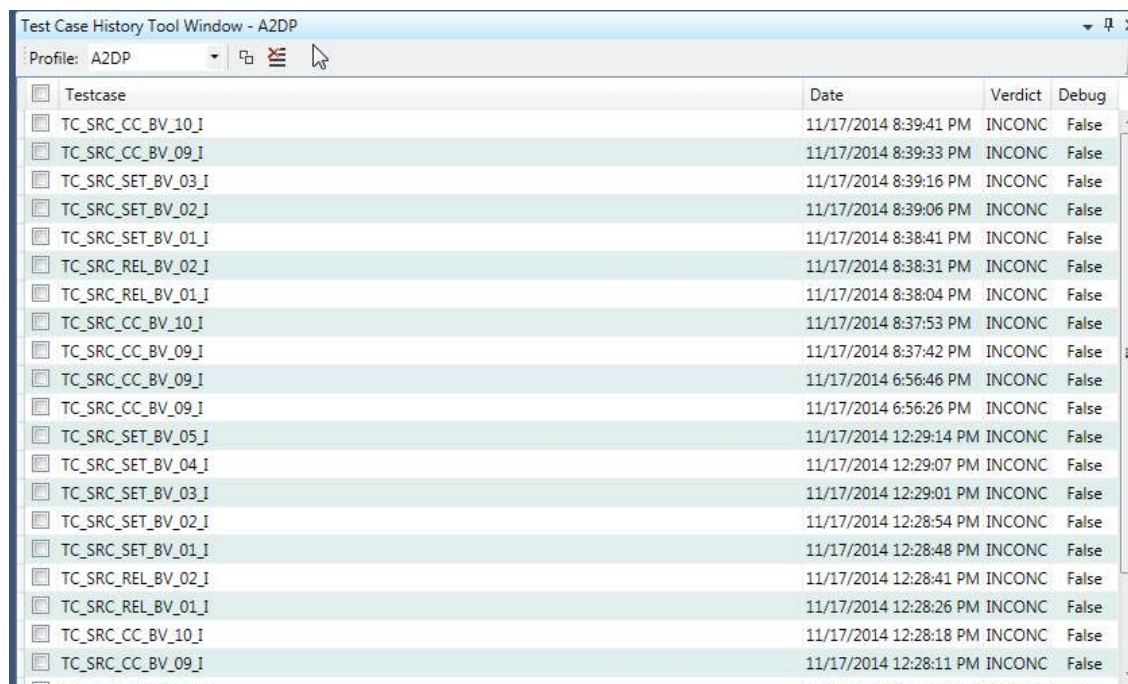
Reviewing and Editing the Test History

Reviewing and Editing the Test History

The complete history of test case executions may be somewhat overwhelming. It might be difficult to separate the results that are needed for a report from the test results that are not. The test history editor can be used to remove results that are no longer of interest.

It should be noted that removing entries from the test history is a permanent operation. Items that have been removed cannot be added to the list later. If a result for a test run is accidentally deleted, then the corresponding test case may need to be executed again in order to have results present for generating a report.

The Test Case History Tool Window displays the test case history for the selected project.



The screenshot shows the 'Test Case History Tool Window - A2DP'. At the top, there is a 'Profile:' dropdown menu set to 'A2DP' and three icons: a square, a red 'X', and a mouse cursor. Below this is a table with four columns: 'Testcase', 'Date', 'Verdict', and 'Debug'. The table contains 20 rows of test case data, all with a verdict of 'INCONC' and a debug status of 'False'. The test cases are listed in descending order of execution time on 11/17/2014.

Testcase	Date	Verdict	Debug
TC_SRC_CC_BV_10_I	11/17/2014 8:39:41 PM	INCONC	False
TC_SRC_CC_BV_09_I	11/17/2014 8:39:33 PM	INCONC	False
TC_SRC_SET_BV_03_I	11/17/2014 8:39:16 PM	INCONC	False
TC_SRC_SET_BV_02_I	11/17/2014 8:39:06 PM	INCONC	False
TC_SRC_SET_BV_01_I	11/17/2014 8:38:41 PM	INCONC	False
TC_SRC_REL_BV_02_I	11/17/2014 8:38:31 PM	INCONC	False
TC_SRC_REL_BV_01_I	11/17/2014 8:38:04 PM	INCONC	False
TC_SRC_CC_BV_10_I	11/17/2014 8:37:53 PM	INCONC	False
TC_SRC_CC_BV_09_I	11/17/2014 8:37:42 PM	INCONC	False
TC_SRC_CC_BV_09_I	11/17/2014 6:56:46 PM	INCONC	False
TC_SRC_CC_BV_09_I	11/17/2014 6:56:26 PM	INCONC	False
TC_SRC_SET_BV_05_I	11/17/2014 12:29:14 PM	INCONC	False
TC_SRC_SET_BV_04_I	11/17/2014 12:29:07 PM	INCONC	False
TC_SRC_SET_BV_03_I	11/17/2014 12:29:01 PM	INCONC	False
TC_SRC_SET_BV_02_I	11/17/2014 12:28:54 PM	INCONC	False
TC_SRC_SET_BV_01_I	11/17/2014 12:28:48 PM	INCONC	False
TC_SRC_REL_BV_02_I	11/17/2014 12:28:41 PM	INCONC	False
TC_SRC_REL_BV_01_I	11/17/2014 12:28:26 PM	INCONC	False
TC_SRC_CC_BV_10_I	11/17/2014 12:28:18 PM	INCONC	False
TC_SRC_CC_BV_09_I	11/17/2014 12:28:11 PM	INCONC	False

Use the drop down menu to select a project:

Selecting individual test case results to be deleted

Specific test case results can be selected for deletion by placing a checkmark in the corresponding checkboxes.

Test Case History Tool Window - A2DP

Profile: A2DP

Testcase	Date	Verdict	Debug
☐ TC_SRC_CC_BV_10_I	11/17/2014 8:39:41 PM	INCONC	False
☒ TC_SRC_CC_BV_09_I	11/17/2014 8:39:33 PM	INCONC	False
☒ TC_SRC_SET_BV_03_I	11/17/2014 8:39:16 PM	INCONC	False
☐ TC_SRC_SET_BV_02_I	11/17/2014 8:39:06 PM	INCONC	False
☒ TC_SRC_SET_BV_01_I	11/17/2014 8:38:41 PM	INCONC	False
☐ TC_SRC_REL_BV_02_I	11/17/2014 8:38:31 PM	INCONC	False
☐ TC_SRC_REL_BV_01_I	11/17/2014 8:38:04 PM	INCONC	False
☒ TC_SRC_CC_BV_10_I	11/17/2014 8:37:53 PM	INCONC	False
☐ TC_SRC_CC_BV_09_I	11/17/2014 8:37:42 PM	INCONC	False
☐ TC_SRC_CC_BV_09_I	11/17/2014 6:56:46 PM	INCONC	False
☐ TC_SRC_CC_BV_09_I	11/17/2014 6:56:26 PM	INCONC	False
☐ TC_SRC_SET_BV_05_I	11/17/2014 12:29:14 PM	INCONC	False
☒ TC_SRC_SET_BV_04_I	11/17/2014 12:29:07 PM	INCONC	False
☐ TC_SRC_SET_BV_03_I	11/17/2014 12:29:01 PM	INCONC	False

Delete the selected test cases by clicking the Removed Checked Items button.



It may be useful to select all of the items in the list, and then remove the selection from the items that are to be kept. The "Select All" checkbox may be used to select all of the items.

Test Case History Tool Window - A2DP

Profile: A2DP

Testcase	Date	Verdict	Debug
☒ TC_SRC_CC_BV_10_I	11/17/2014 8:39:41 PM	INCONC	False
☒ TC_SRC_CC_BV_09_I	11/17/2014 8:39:33 PM	INCONC	False
☒ TC_SRC_SET_BV_03_I	11/17/2014 8:39:16 PM	INCONC	False
☒ TC_SRC_SET_BV_02_I	11/17/2014 8:39:06 PM	INCONC	False
☒ TC_SRC_SET_BV_01_I	11/17/2014 8:38:41 PM	INCONC	False
☒ TC_SRC_REL_BV_02_I	11/17/2014 8:38:31 PM	INCONC	False
☒ TC_SRC_REL_BV_01_I	11/17/2014 8:38:04 PM	INCONC	False
☒ TC_SRC_CC_BV_10_I	11/17/2014 8:37:53 PM	INCONC	False

Selecting older test results for deletion

When multiple results are present for one or more test cases, the Select Duplicates button may be used to select the oldest results for each test case.

It should be noted that using Select Duplicates may inadvertently select test case results that need to be kept. As can be seen in the picture to the right, the use of Select Duplicates resulted in a "FAILED" test case run being unselected, that is, to be kept; while the previous successful execution ("PASS") was marked for deletion. This situation is similar to that which can result when using the "Select Latest" button while selecting test case results to be included in a report. (See section 3.7, "Selecting the test case results to be used in the report".)

Test Case History Tool Window - A2DP

Profile: A2DP

Check duplicate item(s)

Testcase	Date	Verdict	Debug
☒ TC_SRC_CC_BV_09_I	11/17/2014 12:14:48 PM	INCONC	False
☒ TC_SRC_CC_BV_10_I	11/17/2014 12:15:19 PM	INCONC	True
☒ TC_SRC_REL_BV_01_I	11/17/2014 12:15:35 PM	INCONC	False
☐ TC_SRC_SDP_BV_01_I	11/17/2014 12:20:59 PM	INCONC	False
☒ TC_SRC_CC_BV_09_I	11/17/2014 12:28:11 PM	INCONC	False
☒ TC_SRC_CC_BV_10_I	11/17/2014 12:28:18 PM	INCONC	False
☒ TC_SRC_REL_BV_01_I	11/17/2014 12:28:26 PM	INCONC	False
☒ TC_SRC_REL_BV_02_I	11/17/2014 12:28:41 PM	INCONC	False
☒ TC_SRC_SET_BV_01_I	11/17/2014 12:28:48 PM	INCONC	False
☒ TC_SRC_SET_BV_02_I	11/17/2014 12:28:54 PM	INCONC	False
☒ TC_SRC_SET_BV_03_I	11/17/2014 12:29:01 PM	INCONC	False
☐ TC_SRC_SET_BV_04_I	11/17/2014 12:29:07 PM	INCONC	False
☐ TC_SRC_SET_BV_05_I	11/17/2014 12:29:14 PM	INCONC	False
☒ TC_SRC_CC_BV_09_I	11/17/2014 6:56:26 PM	INCONC	False
☒ TC_SRC_CC_BV_09_I	11/17/2014 6:56:46 PM	INCONC	False
☒ TC_SRC_CC_BV_09_I	11/17/2014 8:37:42 PM	INCONC	False
☒ TC_SRC_CC_BV_10_I	11/17/2014 8:37:53 PM	INCONC	False
☐ TC_SRC_REL_BV_01_I	11/17/2014 8:38:04 PM	INCONC	False
☐ TC_SRC_REL_BV_02_I	11/17/2014 8:38:31 PM	INCONC	False
☐ TC_SRC_SET_BV_01_I	11/17/2014 8:38:41 PM	INCONC	False
☐ TC_SRC_SET_BV_02_I	11/17/2014 8:39:06 PM	INCONC	False
☐ TC_SRC_SET_BV_03_I	11/17/2014 8:39:16 PM	INCONC	False
☐ TC_SRC_CC_BV_09_I	11/17/2014 8:39:33 PM	INCONC	False

To view older test cases, sort the test cases by date.

Test Case History Tool Window - A2DP

Profile: A2DP

Testcase	Date	Verdict	Debug
☒ TC_SRC_CC_BV_09_I	11/17/2014 12:14:48 PM	INCONC	False
☒ TC_SRC_CC_BV_10_I	11/17/2014 12:15:19 PM	INCONC	True

Deleting test case results

Once test cases have been selected for deletion, click the Remove Checked Items button.

Note that there is no confirmation for this operation.



Scripting

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.

This document describes "Scripted Operation".

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.

"Fully Automated Operation" is described in the "Extended Automating" reference document.

"Scripted Operation"

Each Project (Test Suite) in a PTS Workspace has a "Test Script" associated with it. The Test Script contains names of Test Cases from the Project in the order in which they should be executed. The Test Cases included in the script can be in any order and can appear multiple times as needed.

To use Scripted Operation:

1. Open a pre-existing PTS Workspace (or, create a new one if needed);
2. Select one of the projects in the Workspace and make it the active project;
3. Select the Test Cases that will be included in the Test Script;
4. Confirm that the ICS and IXIT settings are correct;
5. Execute the script!

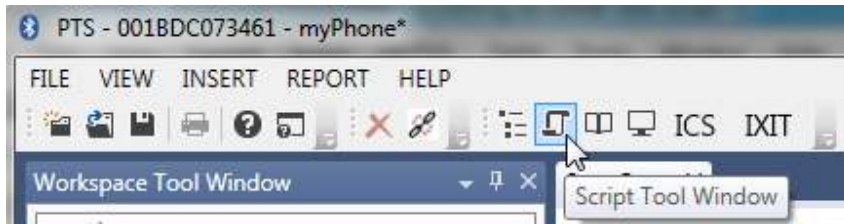
Once the Test Script is complete and ready to be used again in the future, it can be scheduled to automatically execute at a later time. At that time the script can be run multiple times or until a certain amount of time has passed.

Often, you may want to have the script run unattended after normal hours. Combining "Scripted Operation" with "Operator-less Operation", allows for such unattended operation.

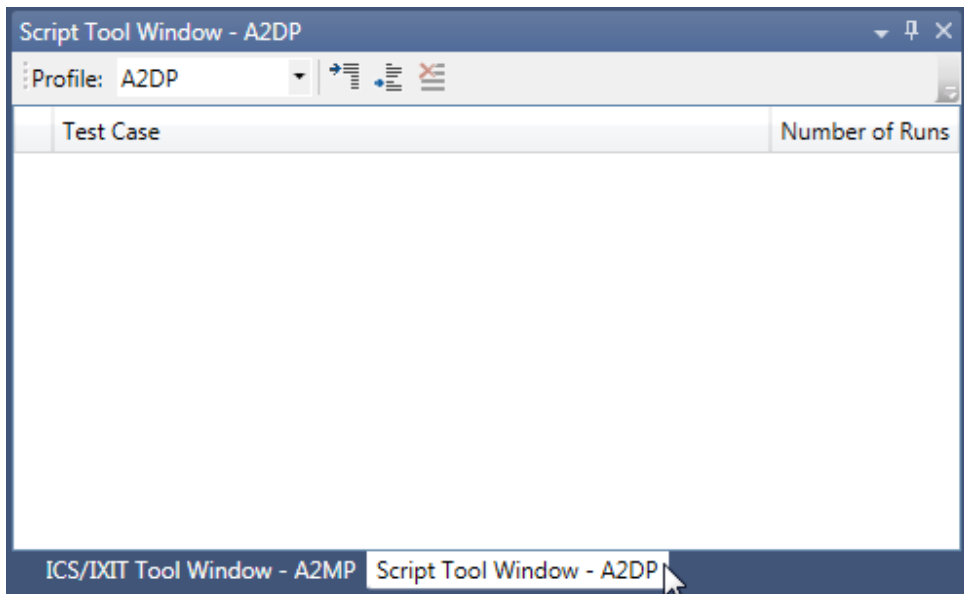
Note that each Project only has one Test Script associated with it. In order to have more than one Test Script for a given Project, create another Workspace that contains the desired Test Suite.

Creating an initial Test Script

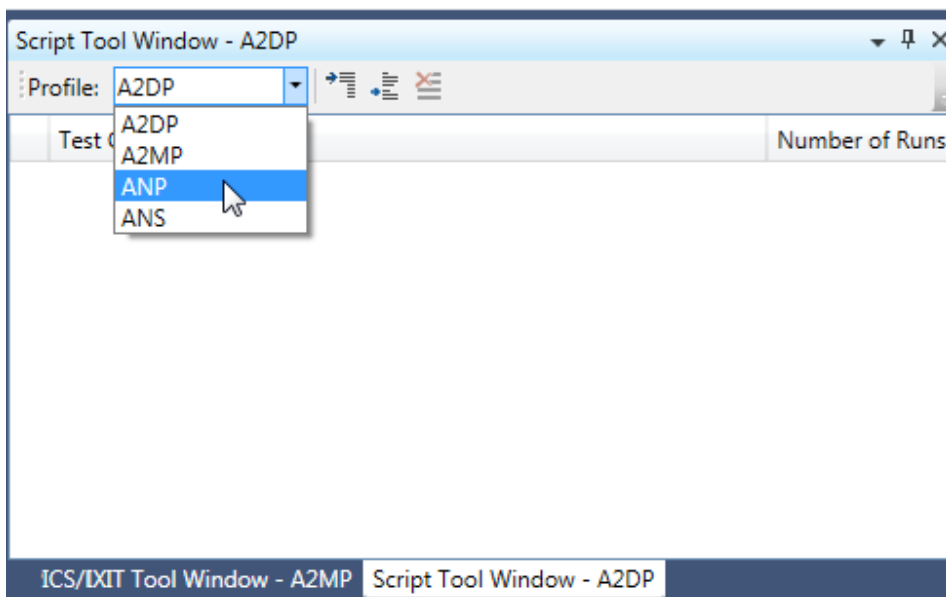
To select the Script Tool Window, click the Script Tool Window icon on the toolbar.



Also, you can switch between the ICS/IXIT Tool Window and the Script Tool Window by clicking on the tab at the bottom of the window.



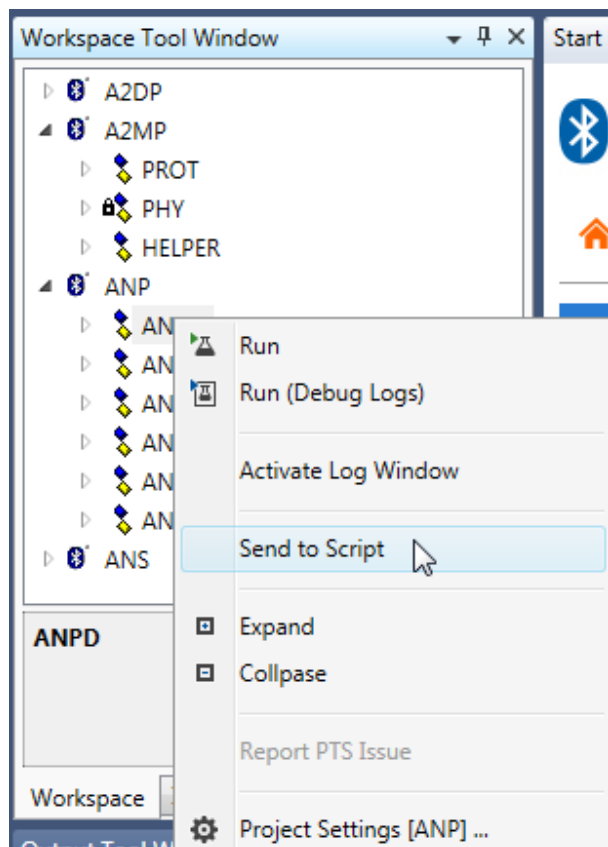
The Test Script for each project is named "Profile Name" Script and will appear in the drop down menu in the Script Tool Window.



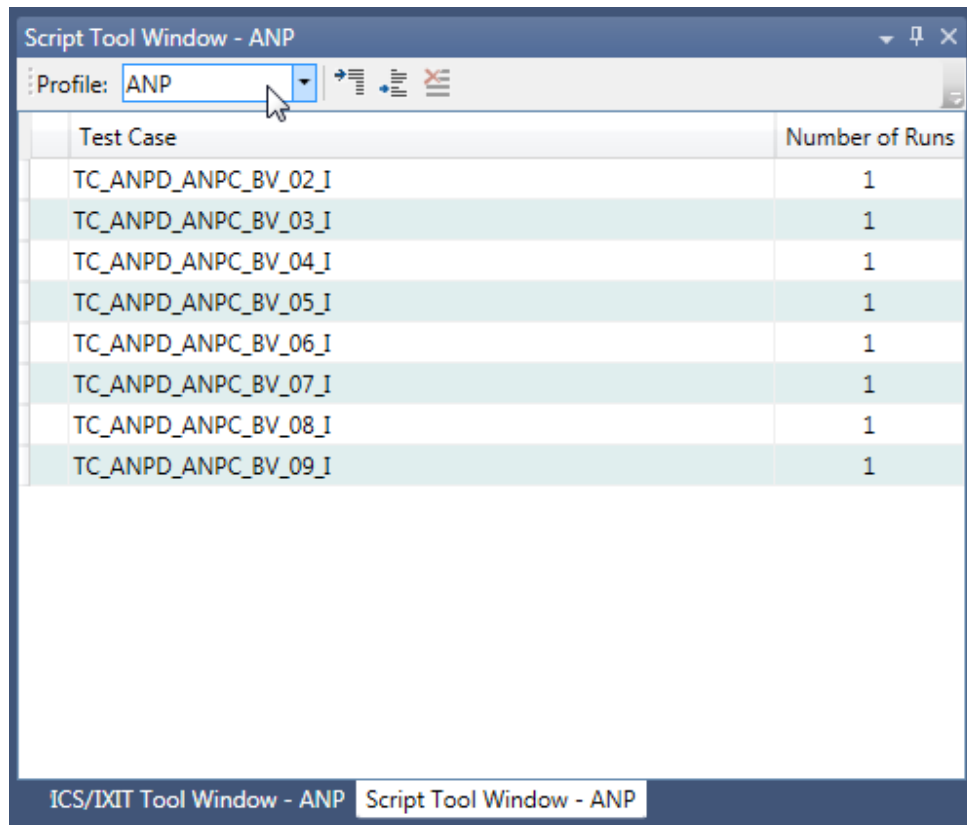
In this document, the Test Script tab and/or window will be referred to as the Script Tool Window.

Adding Test Cases to the Test Script

Choose a Test Case from the Test Case tree in the Workspace window and right click on it. Select "Send to Test Script" and the Test Case will be added to the "Script Tool Window" for the selected Test Suite.



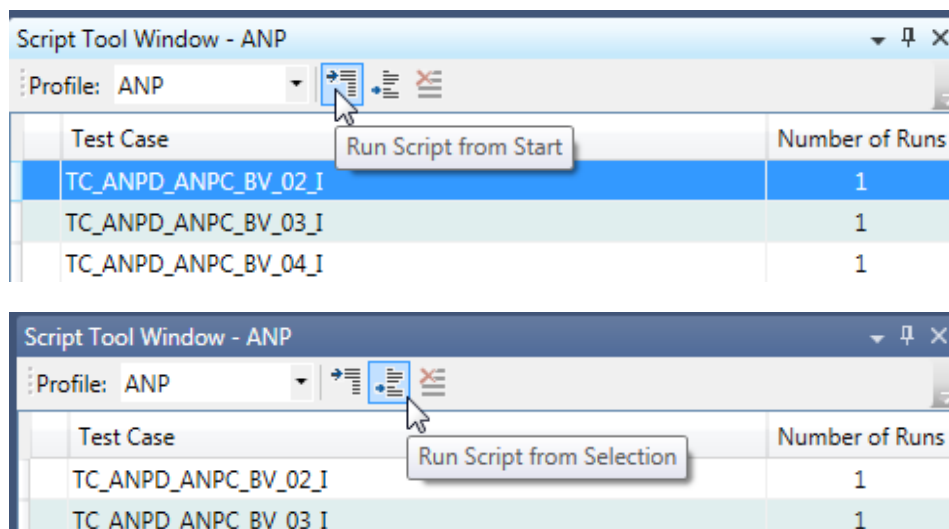
Repeat the process until all of the desired Test Cases appear in the "Script Tool Window". The Test Cases can be selected in any order and may be selected as many times as needed.



Executing a Test Script

Once the desired set of Test Cases has been entered into the Test Script it is time to execute it.

To execute a script click the "Run Script" icon or "Run Script from selection" icon.



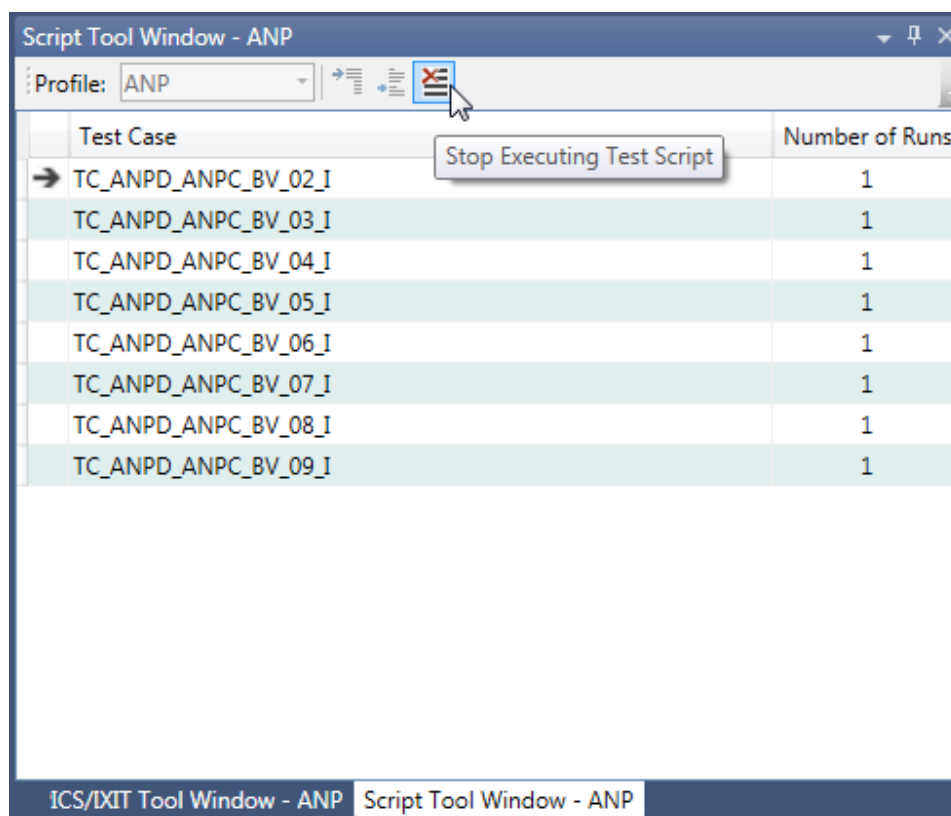
The first two items in the "Script" menu control the execution of the script:

- "Run from Start" – Executes the Test Script starting at the first Test Case listed, proceeding in order through the tests until the last Test Case is run.
- "Run from Selection" – If a Test Case is currently selected in the "Test Script Window", this item will be enabled. Selecting "Run from Selection" will start the script at the selected Test Case, proceeding in order through the tests until the last Test Case is run.

Stopping Test Script execution

When a Test Script is executing, the "Stop executing test script" button will be enabled in the Script Tool Window. When pressed, the button will

1. Terminate the currently running Test Case;
2. Terminate the execution of the Test Script.



Editing a Test Script

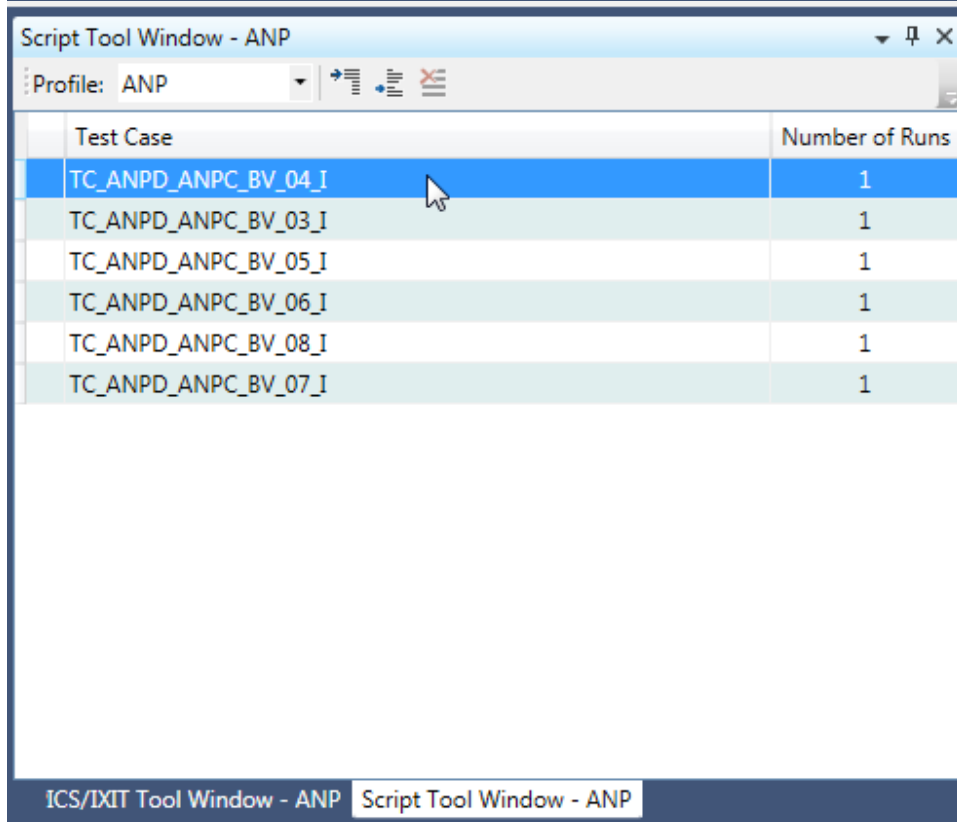
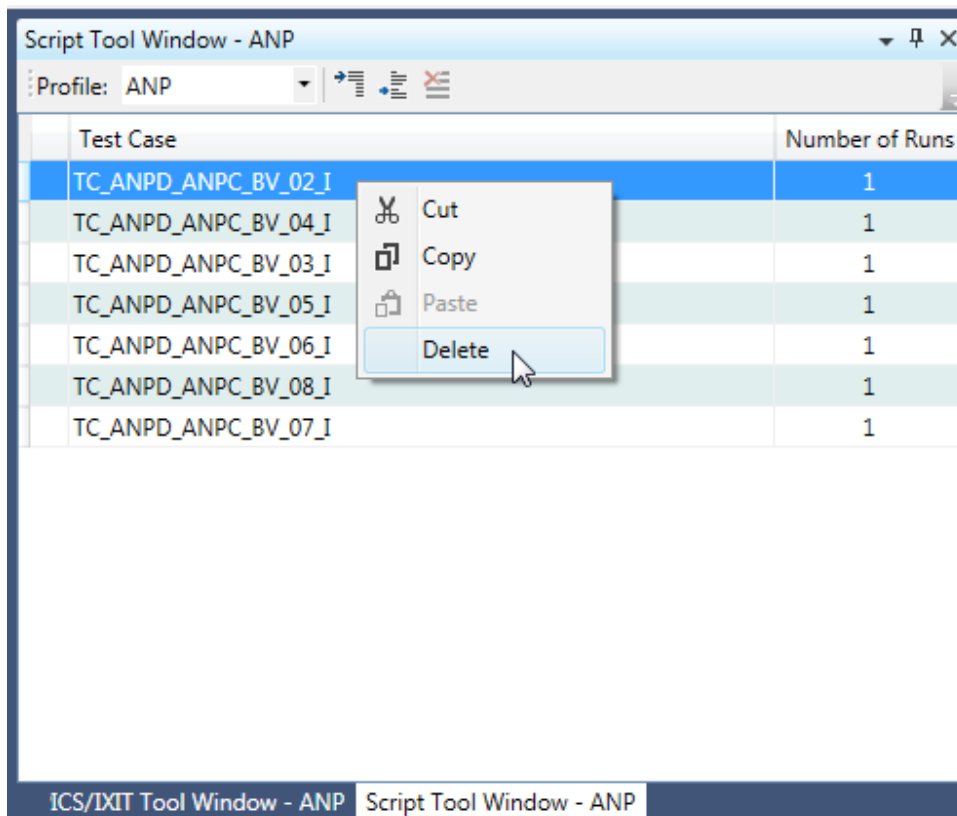
Editing a Test Script

Once the desired list of Test Cases has been added to the Test Script, the script may be edited to remove

Test Cases or change the order in which the Test Cases are executed.

Removing a Test Case from the Test Script

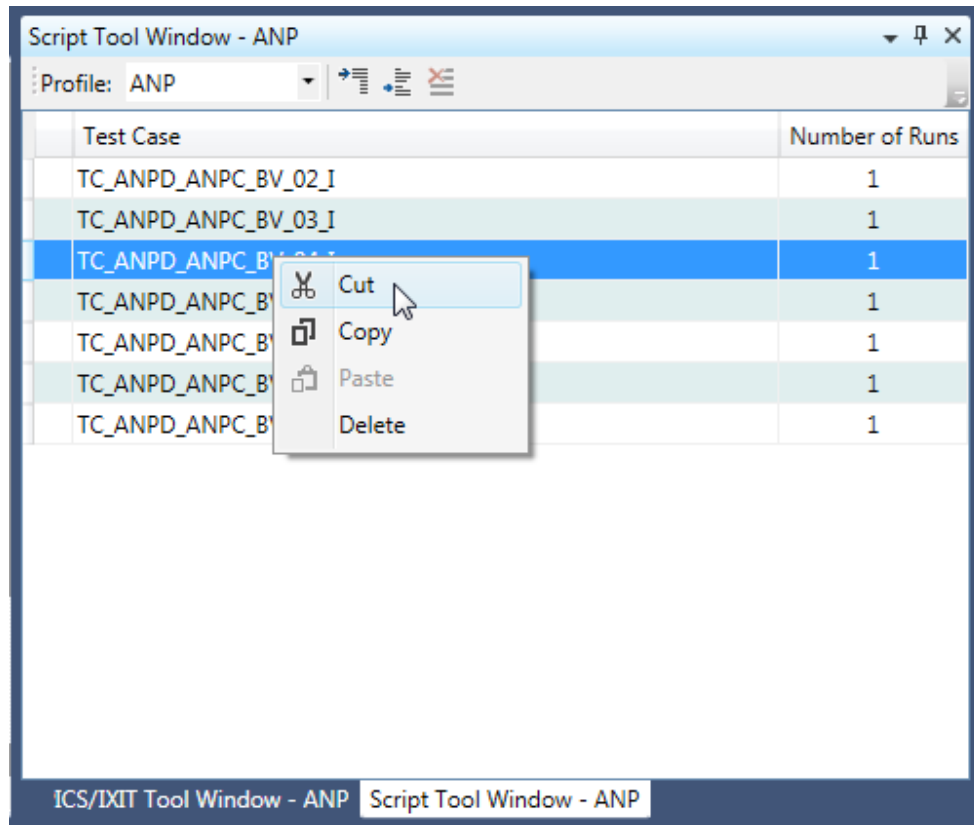
In the Script Tool Window, highlight the Test Case that is to be removed and right click on it. Select the "Delete" item from the popup menu and the Test Case will be removed from the script.



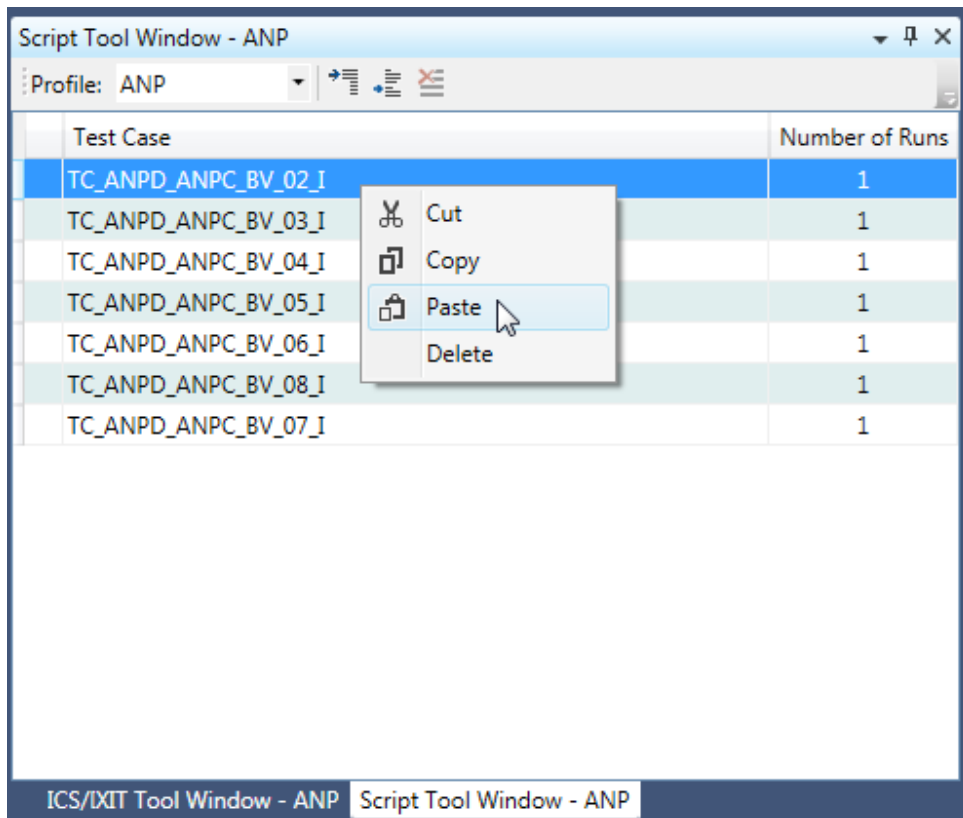
Changing the order of execution

The Test Script is executed starting with the first Test Case in the script and proceeds sequentially until the last test has been performed. The order of execution may be changed using the following steps:

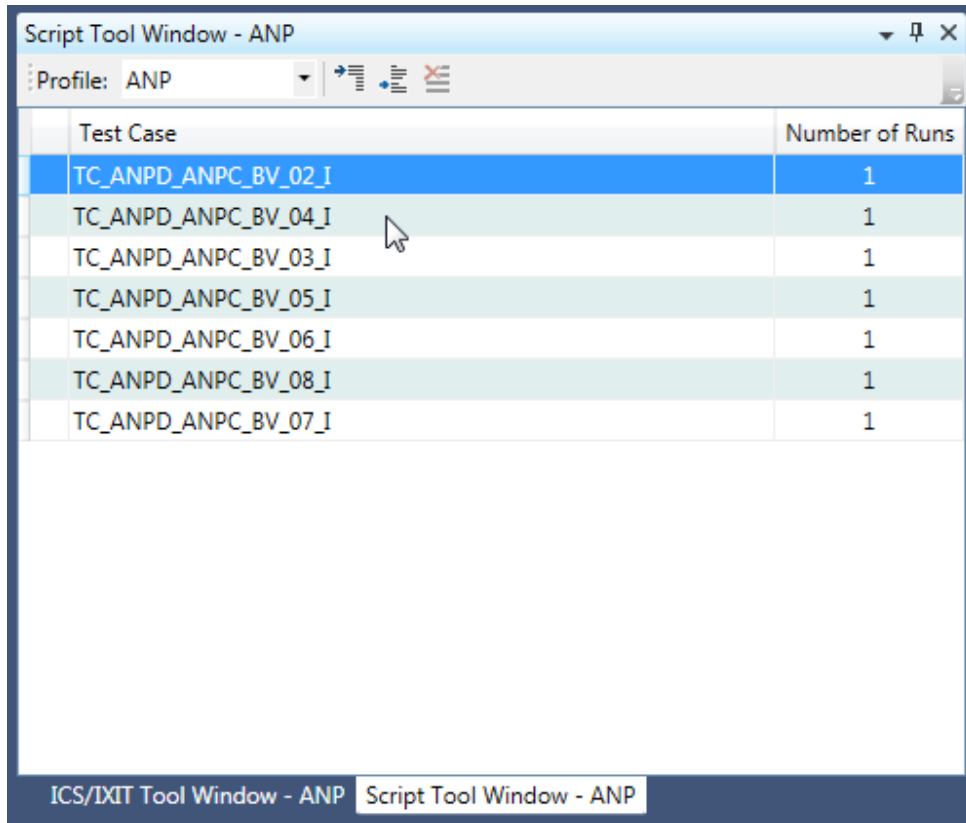
1. Highlight a Test Case that should execute at a different place in the script in the "Test Script Window";
2. Right click on the test and select "Cut" from the popup menu;



3. Highlight the Test Case before position you want to move the "Cut" Test case to:



4. Right click on this second test and select paste:



The screenshot shows a window titled "Script Tool Window - ANP". At the top, there is a "Profile:" dropdown menu set to "ANP" and three icons: a list, a plus sign, and a minus sign. Below this is a table with two columns: "Test Case" and "Number of Runs". The table contains seven rows of test cases, each with a value of 1 in the "Number of Runs" column. A mouse cursor is hovering over the second row, "TC_ANPD_ANPC_BV_04_I". At the bottom of the window, there are two tabs: "ICS/IXIT Tool Window - ANP" and "Script Tool Window - ANP", with the latter being the active tab.

Test Case	Number of Runs
TC_ANPD_ANPC_BV_02_I	1
TC_ANPD_ANPC_BV_04_I	1
TC_ANPD_ANPC_BV_03_I	1
TC_ANPD_ANPC_BV_05_I	1
TC_ANPD_ANPC_BV_06_I	1
TC_ANPD_ANPC_BV_08_I	1
TC_ANPD_ANPC_BV_07_I	1

Adding a Test Case to the Test Script

There are two ways to add an additional Test Case to the script.

The first method is to select a Test Case and add it to the script using the "Send to Test Script" popup menu item described above in [Adding Test Cases to the Test Script](#). The Test Case will be added at the end of the script, at which time it can be moved to its desired execution point using the steps in the previous section (4.2, "Changing the order of execution").

The second method, if the Test Case is already in the script, is to follow the instructions in the previous section, but select "Copy" instead of "Cut".

Logging

Introduction

The Profile Tuning Suite (PTS) can produce three types of output during the execution of a test case:

- A log of the test case execution;
- A protocol trace via the PTS Protocol Viewer.

Each of these contains basically the same information. They vary in the way in which the information is presented and in the amount of detail that may be present.

PTS contains test suites for both “traditional” Bluetooth BR/EDR and for Bluetooth Low Energy (LE). The two types have different requirements at the application level, so their test suites are created using different development environments.

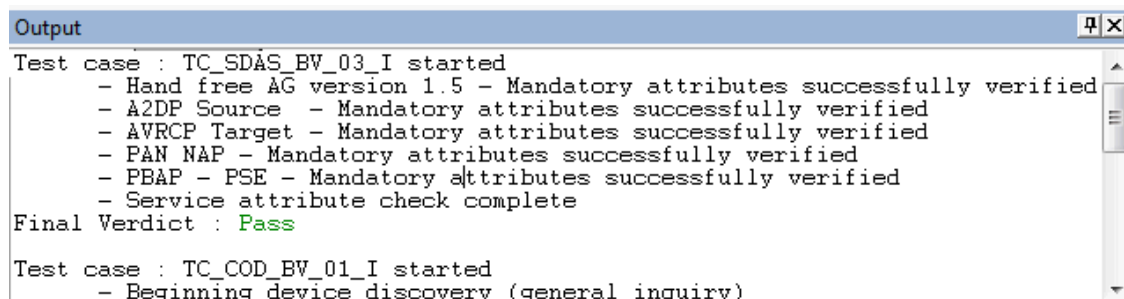
PTS BR/EDR test cases are developed using a test scripting language known as TTCN (Testing and Test Control Notation1.) The TTCN operating environment provides logging support for over two dozen different events that may occur during the execution of a test. PTS uses this logging support to create text case execution logs.

Bluetooth Low Energy profiles focus less on the messaging between devices and more on the XML based data that is being exchanged. For this reason, the PTS LE test suites are written in C++. There is much less execution log information available because there are a smaller number of unique events to be logged.

Low level capture of the communications between PTS and the Implementation Under Test can be achieved using the PTS Protocol Viewer (PTSPV). A software “tap point” between the PTS software (Bluetooth Host) and the PTS endpoint device (Bluetooth Controller) captures all of the communications between the devices and forwards the captured data to the Protocol Viewer application.

Output Window

The Output window in the lower left hand corner of the PTS display contains a summary of test case executions for the current session of PTS. During the execution of a test case, information and status messages known as “Verdict Descriptions” are output to the test case execution log. Each “Verdict Description” that is output to the log will also appear in the Output window along with the final verdict of the test case.

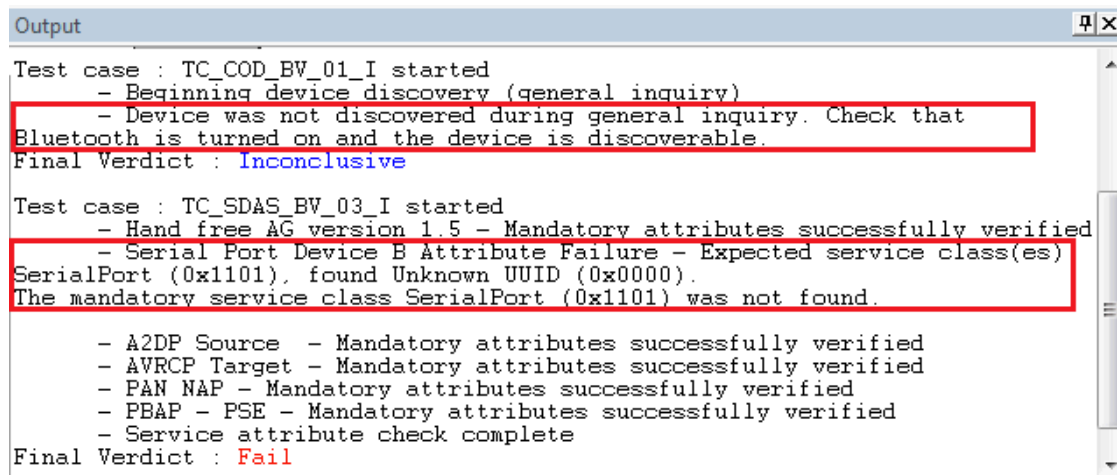


```
Output
Test case : TC_SDAS_BV_03_I started
- Hand free AG version 1.5 - Mandatory attributes successfully verified
- A2DP Source - Mandatory attributes successfully verified
- AVRCP Target - Mandatory attributes successfully verified
- PAN NAP - Mandatory attributes successfully verified
- PBAP - PSE - Mandatory attributes successfully verified
- Service attribute check complete
Final Verdict : Pass
Test case : TC_COD_BV_01_I started
- Beginning device discovery (general inquiry)
```

The Output window is the first place to look when there is a problem during the execution of a test. When a problem occurs, a description of the problem will be output to the execution log along with the Output window.

The descriptive text is identical to the text in the test execution log. Searching for that text in the log output makes it possible to quickly locate the area where the problem occurred.

The



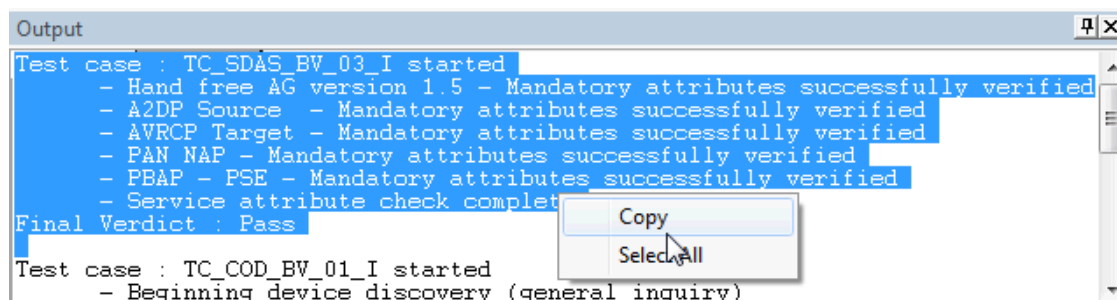
```

Output
Test case : TC_COD_BV_01_I started
- Beginning device discovery (general inquiry)
- Device was not discovered during general inquiry. Check that
Bluetooth is turned on and the device is discoverable.
Final Verdict : Inconclusive

Test case : TC_SDAS_BV_03_I started
- Hand free AG version 1.5 - Mandatory attributes successfully verified
- Serial Port Device B Attribute Failure - Expected service class(es)
SerialPort (0x1101), found Unknown UUID (0x0000).
The mandatory service class SerialPort (0x1101) was not found.

- A2DP Source - Mandatory attributes successfully verified
- AVRCP Target - Mandatory attributes successfully verified
- PAN NAP - Mandatory attributes successfully verified
- PBAP - PSE - Mandatory attributes successfully verified
- Service attribute check complete
Final Verdict : Fail
  
```

The contents of the Output window are not saved when PTS is terminated. Information in the window can be highlighted using the keyboard or mouse in the normal way. Once an area of text has been selected, it may be copied to the Windows clipboard by right clicking in the Output window. The text may then be pasted into a word processor or text editor.



```

Output
Test case : TC_SDAS_BV_03_I started
- Hand free AG version 1.5 - Mandatory attributes successfully verified
- A2DP Source - Mandatory attributes successfully verified
- AVRCP Target - Mandatory attributes successfully verified
- PAN NAP - Mandatory attributes successfully verified
- PBAP - PSE - Mandatory attributes successfully verified
- Service attribute check complete
Final Verdict : Pass

Test case : TC_COD_BV_01_I started
- Beginning device discovery (general inquiry)
  
```

Test Case History Tool Window

The Test Case History window is in the lower right hand corner of PTS display. This window contains a one line summary of each test case that has been executed in the currently open workspace. The summary contains

- The name of the test suite.
- The name of the test case.
- The date and time that the test case was executed.
- The final verdict of the test case execution.

Test Case History				
DID	TC_SDI_BV_1_I	Date: Tuesday, August 21, 2012 21:03:45	Verdict : PASS	[D]
DID	TC_SDI_BV_1_I	Date: Tuesday, August 21, 2012 21:03:05	Verdict : FAIL	
DID	TC_SDI_BV_1_I	Date: Tuesday, August 21, 2012 21:02:25	Verdict : FAIL	
IOPT	TC_SDSS_BV_02_I	Date: Tuesday, August 21, 2012 21:10:53	Verdict : PASS	[D]
IOPT	TC_SDSS_BV_02_I	Date: Tuesday, August 21, 2012 21:05:03	Verdict : FAIL	[D]
IOPT	TC_COD_BV_01_I	Date: Tuesday, August 21, 2012 21:04:12	Verdict : PASS	

Left clicking on an entry in this window will cause the corresponding test case execution log to be displayed in the logfile.log window.

Right clicking on an entry will display "Show in Explorer". Selecting "Show in Explorer" will cause a Windows Explorer window to be opened.

Test Case History				
DID	TC_SDI_BV_1_I	Date: Tuesday, August 21, 2012 21:03:45	Verdict : PASS	[D]
DID	TC_SDI_BV_1_I	Date: Tuesday, August 21, 2012 21:03:05	Verdict : FAIL	
DID	TC_SDI_BV_1_I	Date: Tuesday, August 21, 2012 21:02:25	Verdict : FAIL	
IOPT	TC_SDSS_BV_02_I	Date: Tuesday, August 21, 2012 21:10:53	Verdict : PASS	[D]
IOPT	TC_SDSS_BV_02_I	Date: Tuesday, August 21, 2012 21:05:03	Verdict : FAIL	[D]
IOPT	TC_COD_BV_01_I	Date: Tuesday, August 21, 2012 21:04:12	Verdict : PASS	

If the entry ends in a "[D]", then the corresponding test execution was under the control of the "Run (Debug Logs)" feature described in [Executing a single test case](#). In this case the folder containing all of the relevant logfiles will be displayed.

If the entry does not end in a "[D]", then the main folder for the project will be displayed with the test case execution log selected.

Test Execution Log

Test Execution Log

The large window in the center of the PTS display contains tabs for a number of items. One of those items, the test case execution log, is found on the tab labeled "Test Case" Log.



The log file tab contains only one test case execution log at a time. The contents of the log are cleared at the start of the test case and only the events that occur during that run will be present in the display. A previous test case log may be recalled at any time, see [Test Case History Tool Window](#), for more information.

It should be noted that there may be more than one execution log tab shown in the display. If test cases are executed from more than one project, then there will be an execution log for each project. The same holds true when reviewing previous execution logs from more than one project.

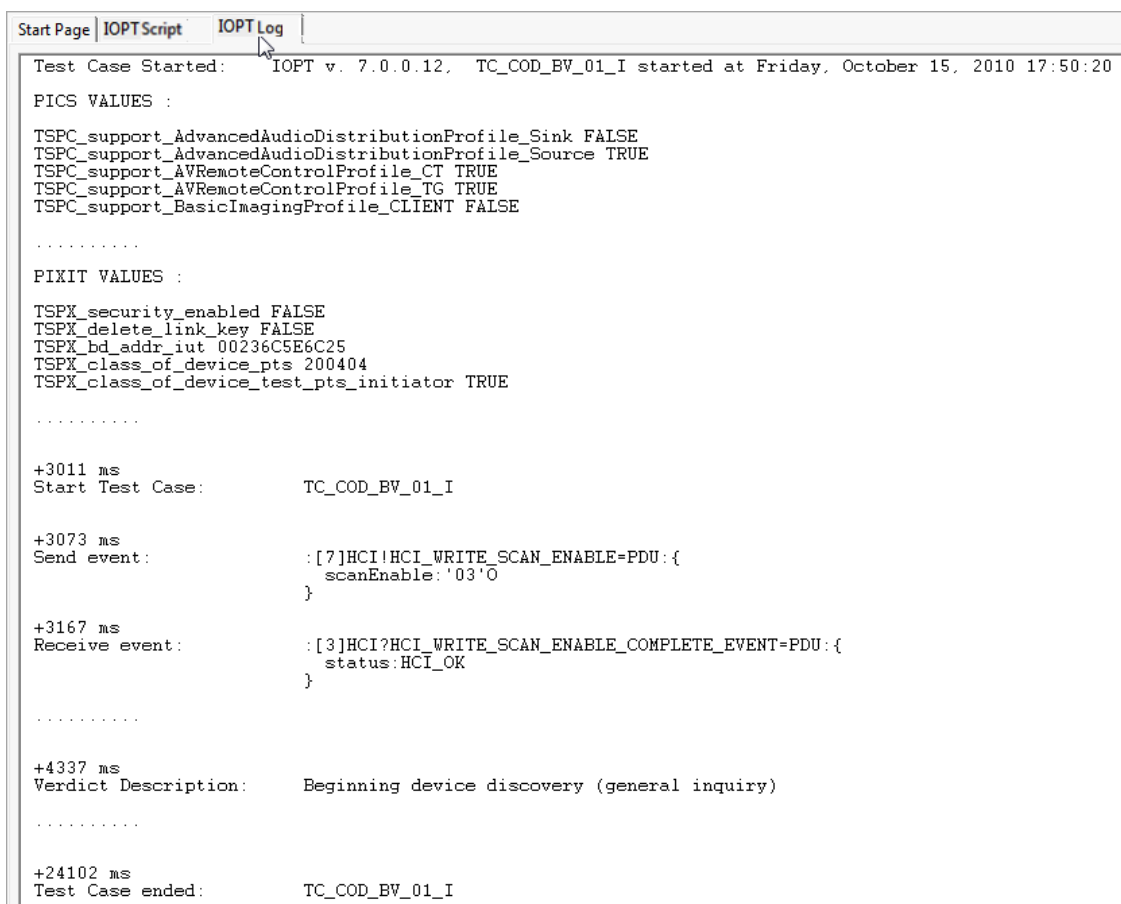


Format of the execution log

The first line of the log, labeled "Test Case Started", contains

- the name of the test suite;
- the version number for the test suite executable;
- the name of the test case being executed;
- the date and time at which the test began.

This information is followed by the list of ICS values used by the test suite and their settings. The IXIT values for the test suite are listed next.



The screenshot shows a software window titled 'IOPT Log' with tabs for 'Start Page', 'IOPT Script', and 'IOPT Log'. The log content is as follows:

```

Test Case Started:      IOPT v. 7.0.0.12, TC_COD_BV_01_I started at Friday, October 15, 2010 17:50:20

PICS VALUES :

TSPC_support_AdvancedAudioDistributionProfile_Sink FALSE
TSPC_support_AdvancedAudioDistributionProfile_Source TRUE
TSPC_support_AVRemoteControlProfile_CT TRUE
TSPC_support_AVRemoteControlProfile_TG TRUE
TSPC_support_BasicImagingProfile_CLIENT FALSE
.....

PIXIT VALUES :

TSPX_security_enabled FALSE
TSPX_delete_link_key FALSE
TSPX_bd_addr_iut 00236C5E6C25
TSPX_class_of_device_pts 200404
TSPX_class_of_device_test_pts_initiator TRUE
.....

+3011 ms
Start Test Case:      TC_COD_BV_01_I

+3073 ms
Send event:           :[7]HCI!HCI_WRITE_SCAN_ENABLE=PDU:{
                        scanEnable:'03'0
                        }

+3167 ms
Receive event:        :[3]HCI?HCI_WRITE_SCAN_ENABLE_COMPLETE_EVENT=PDU:{
                        status:HCI_OK
                        }
.....

+4337 ms
Verdict Description:   Beginning device discovery (general inquiry)
.....

+24102 ms
Test Case ended:      TC_COD_BV_01_I

```

Test case events are then added to the log as they occur. Each event is displayed in a two column format where the first column contains a timestamp and the name of the event such as “Start Test Case”, “Verdict Description”, “Send event”, etc. The timestamp is in milliseconds relative to the start time shown in the first line of the log.

The second column contains information about the event. In the case of a “Verdict Description” this will be the summary text that is also sent to the Output window.

The information displayed for an event is particular to the event itself. For example, for events such as “Send event” and “Receive event”, the second column will contain the name of the message that is being sent or received, along with the parameters for the message.

Since there are over two dozen different events that may occur during the execution of a test case, the execution log may be difficult to read. See [Selecting the events to be logged](#), for ways to select the events that will be included in the log.

Interesting events

The following events are those that are of general interest to most PTS users.

“Start Test Case”

This event indicates that the execution of the TTCN test script has started.

```
+3011 ms
Start Test Case:      TC_COD_BV_01_I
```

The first line of the test case execution log shows the time that the test case itself started. Initialization of the test environment, display of the ICS and IXIT values, and a slight delay occur before the TTCN test script actually begins.

“Test Case ended”

“Test Case ended” is the last entry in the execution log for a test case. All of the events that occur between “Start Test Case” and “Test Case ended” are a part of the execution of the test script.

```
+24102 ms
Test Case ended:      TC_COD_BV_01_I
```

“Send event”/“Receive event”

“Send” and “Receive” events are used to display communication messages between PTS and an IUT. A “Send event” indicates a protocol message sent from PTS to the IUT, while a “Receive event” shows a message sent from the IUT to PTS.

```
+10639 ms
Send event:      :[7]OBEX!OBEX_CONNECT_REQ=PDU:{
                  bdAddr:'777777777777'O,
                  rfcomm_channel:13,
                  obex_connection_id:'00000000'O,
                  version_number:OMIT,
                  flags:OMIT,
                  max_packet_length:OMIT,
                  headers:[
                    {
                      hi:OBEX_TARGET_e,
                      hv:[
                        target:'796135F0F0C511D809660800200C9A66'O
                      ]
                    }
                  ]
                }

+11716 ms
Receive event:   :[61]OBEX?OBEX_CONNECT_RSP=PDU:{
                  obex_connection_id:'E0CD610D'O,
                  bdAddr:'777777777777'O,
                  resp_code:OBEX_SUCCESS_FINAL_e,
                  version_number:'10'O,
                  flags:'00'O,
                  max_packet_length:'0FA0'O,
                  headers:[
                    {
                      hi:OBEX_CONNECTION_ID_e,
                      hv:[
                        connection_id:'00155F68'O
                      ]
                    },
                    {
                      hi:OBEX_WHO_e,
                      hv:[
                        who:'796135F0F0C511D809660800200C9A66'O
                      ]
                    }
                  ]
                }
```

These events contain three items of interest:

- The protocol layer being used for the message. This will be the indicated immediately to the left of the "!" (send) or "?" (receive) characters. ("OBEX" in the example to the right.)
- The name of the message. (e.g. "OBEX_CONNECT_REQ".)
- The parameters that were sent or received in the message.

The example shows an OBEX connection request being sent from the PTS to a Phone Book Access Profile (PBAP) server on the IUT. The "Receive event" is the response from the server accepting the connection request.

"Verdict Description"

As mentioned above ([Output Window](#)), "Verdict Description" events are used to indicate status, test case state, or other important information. The text for a "Verdict Description" event is always displayed in the Output Window.

```
+4337 ms  
Verdict Description:      Beginning device discovery (general inquiry)
```

The presence of "Verdict Description" events in the test case execution log cannot be disabled.

"Preliminary Verdict"

At various times during the execution of a test case a "Preliminary Verdict" may be issued. This allows the execution of the test to be broken into segments, where each segment stands on its own. The "Preliminary verdict" may be used in determining the final result of the test case.

```
+3183 ms  
Preliminary verdict:      : [3] (PASS)
```

Searching a test execution log for "Preliminary Verdict" events is another way (in addition to searching for "Verdict Descriptions") to identify where a problem may have occurred during a test.

For more information on "Preliminary Verdict", see section 8 ("Verdict Determination").

"Final Verdict"

The "Final Verdict" is the overall result of the test case execution.

```
+24024 ms  
Final verdict:          B: [11] PASS
```

A final verdict will always be displayed in the Output window, even if the "Final Verdict" event is not selected for display in the execution log.

"Encrypted Verdict"

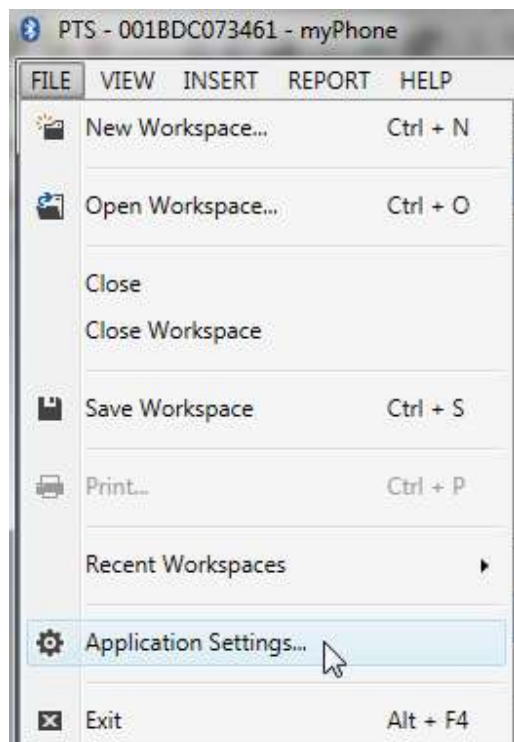
A summation of the test case execution is encrypted for reporting purposes and output to the test case execution log as an "Encrypted Verdict". The information cannot be changed, so it is used as the primary evidence that a test case execution concluded successfully when reporting test results for qualification purposes.

```
+24071 ms  
Encrypted Verdict      A1  
#ZY4MzQ5MDg3MmQwNTIzOGQzMzQ5YjY2ODAwMzNjQ5ZDY4NmY0Yjc2MzIzM2VkNTUyNTFhNGU4Mjg1MjI0NA==#Np9ossP1nS0  
II/fppwPoXPA8gmQilTg2kkzV1tpcw/qX+ea4SC+u+n
```

The presence of "Encrypted Verdict" events in the test case execution log cannot be disabled.

Selecting the events to be logged

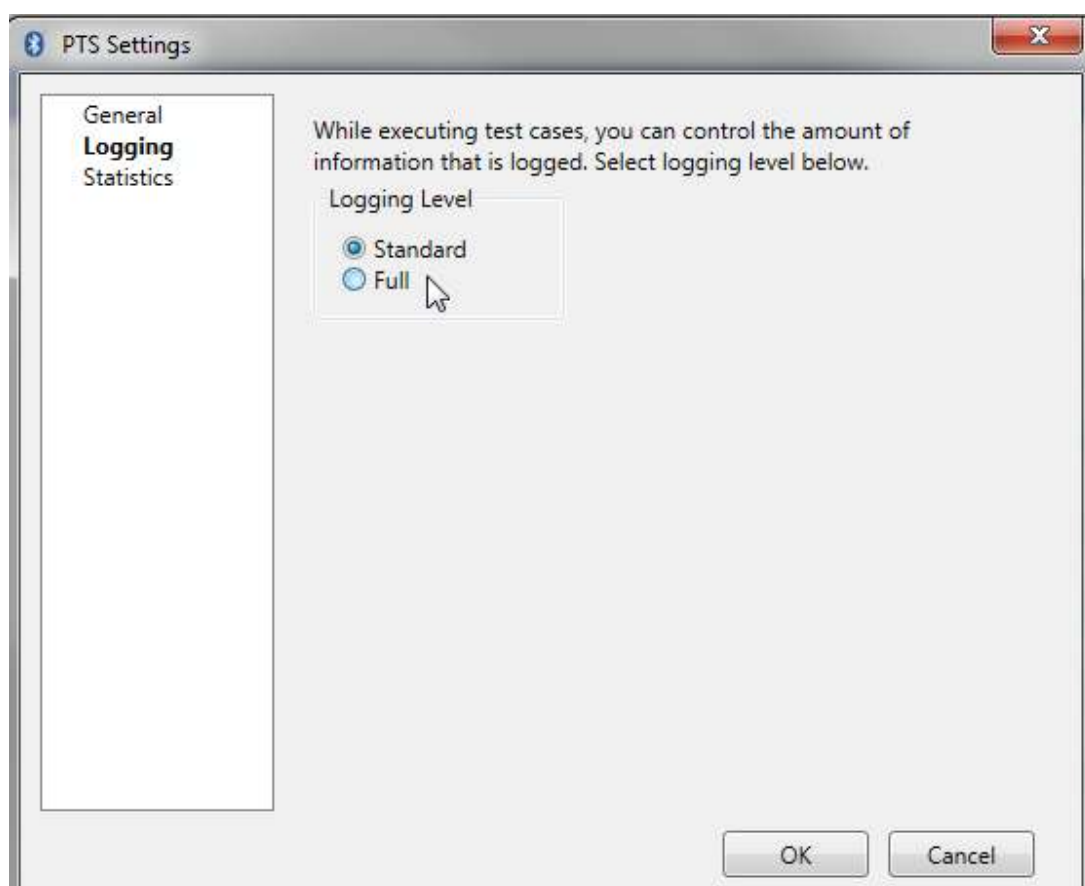
The types of events to be included in the test execution log are selected via the "Logging" page of the PTS application settings. The application settings are accessed using the "Application Settings" item on the "File" menu.



As mentioned in [Executing a single test case](#), changes to the log level may never be needed. The “Run (Debug Logs)” features is a much better way to produce highly detailed logs for problem analysis or archival purposes.

The “Logging” page allows for the selection of one of two preset collections of events, or for a completely custom selection of the events of interest.

- The “Standard” preset adds “Send event” and “Receive event” to the “Minimal” setting, along with other events (such as “Preliminary verdict”) that can be used to determine the final verdict of the test case.
- “Full” causes highly detailed information about the inner workings of a test case to be included in the execution log. This is a lot of information and may be overwhelming to most users.



Members of the PTS Technical Support or Development Teams may ask for “Full” logging when a problem occurs with a test case. The highly detailed execution log, combined with the low level communications trace produced by the PTS Protocol Viewer can be invaluable in attempting to determine why a test case is behaving in a certain way.

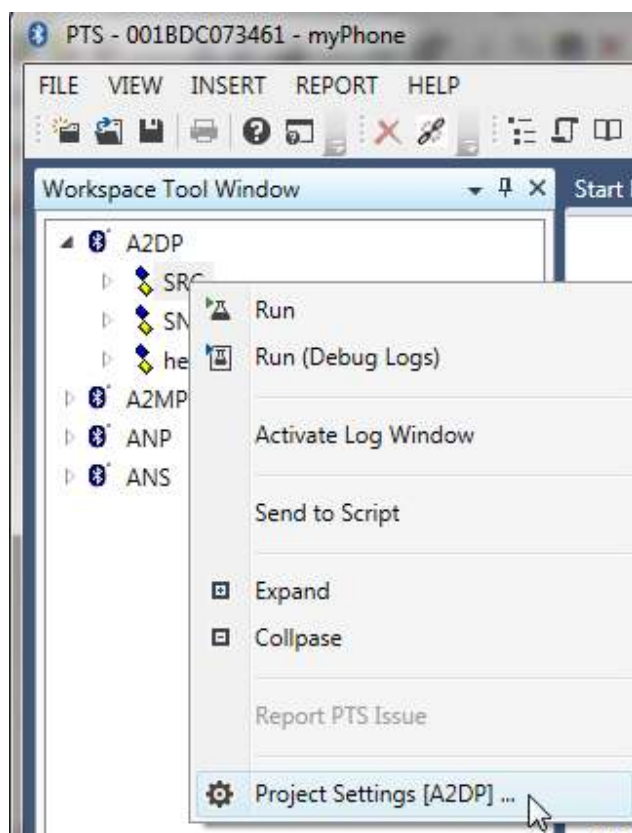
As mentioned in [Introduction](#), Bluetooth Low Energy test suites have a much smaller number of events that can be logged when compared to BR/EDR test suites. In fact, for the LE test suites, there is no difference between “Standard” and “Full”.

“Run-time” vs. deferred logging

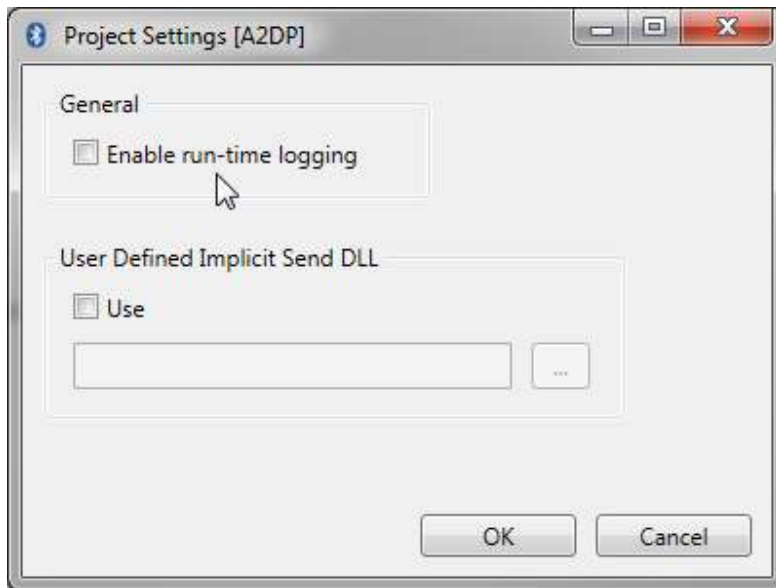
Normally events are added to the test case execution log as they occur. Sometimes this “live” update can have a serious impact on the performance of test execution. In some cases this may result in communication timing changes leading to unexpected results.

If this happens, then the live or “run-time” logging should be disabled. When “run-time” logging is disabled, the information to be displayed in the log is cached until the test case terminates. At that time, the information will be output to the display.

Enabling or disabling live update is selected on the “General” page of “Project Settings.” Right click on a project name at the top level of the “Test Case View” followed by the selection of “Settings...” from the popup menu.



The checkbox labeled “Enable run-time logging” enables or disables the live update feature.



PTS Protocol Viewer

PTS Protocol Viewer

The PTS Protocol Viewer (PTSPV) is a companion application that is run concurrently with PTS. Its purpose is to capture the communications between PTS and an Implementation Under Test at the Bluetooth protocol stack level. Each message that passes between the two devices is then decoded in detail at each layer of the Bluetooth stack.

The Protocol Viewer "stack"

The packets sent from PTS to the Protocol Viewer are encapsulated in a small extension to the normal Bluetooth stack. This extension allows for two types of information to be displayed by the Protocol Viewer.

The first layer in the extended stack is the "Virtual Sniffer" layer. The purpose of this layer is to identify whether the packet contains captured communications data or PTS specific information. Packets that are tagged as PTS information are forwarded to the "PTS" layer for decoding.

Frame#	PTS Times...	Type	Test Case	Verdict	Fram...	Delta	Timestamp
79	0	ACL Data			22	00:00:00.0...	11/5/2010 8:50:53.80200...
80	0	ACL Data			27	00:00:00.2...	11/5/2010 8:50:54.07600...
81	0	HCI Event			7	00:00:00.0...	11/5/2010 8:50:54.08100...
82	0	ACL Data			220	00:00:00.0...	11/5/2010 8:50:54.10100...
83	0	PTS Message			52	00:00:00.3...	11/5/2010 8:50:54.44200...
84	0	ACL Data			21	00:00:00.2...	11/5/2010 8:50:54.67700...
85	0	HCI Event			7	00:00:00.0...	11/5/2010 8:50:54.68200...
86	0	ACL Data			22	00:00:00.0...	11/5/2010 8:50:54.70300...
87	0	ACL Data			27	00:00:00.3...	11/5/2010 8:50:55.01200...
88	0	HCI Event			7	00:00:00.0...	11/5/2010 8:50:55.01700...
89	0	ACL Data			150	00:00:00.0...	11/5/2010 8:50:55.04000...
90	0	PTS Message			55	00:00:00.3...	11/5/2010 8:50:55.36000...
91	0	PTS Message			32	00:00:00.1...	11/5/2010 8:50:55.51500...
92	0	Test Case Ended	TC_SDAS_BV_03_I	PASS	15	00:00:02.2...	11/5/2010 8:50:57.80000...

Currently packets in the PTS layer consist of "Test Case Started", "Test Case Ended", and all "Verdict Description" events. The presence of these packets provides a convenient way to correlate a Protocol Viewer "trace" with the information displayed in the test case execution log.

Frame#	PTS Time	Type	Message	Verdict	Frame Size	Delta	Timestamp
15	0	Test Case Started	TC_SDAS_BV_03_I		15		11/5/2010 8:50:44.40600...
62	0	PTS Message	Hand free AG version 1.5 - Mandatory attributes successfully verified	PASS	69	00:00:07.1...	11/5/2010 8:50:51.52800...
69	0	PTS Message	A2DP Source - Mandatory attributes successfully verified	PASS	57	00:00:01.0...	11/5/2010 8:50:52.57300...
76	0	PTS Message	AVRCP Target - Mandatory attributes successfully verified	PASS	57	00:00:00.8...	11/5/2010 8:50:53.48200...
83	0	PTS Message	PAN NAP - Mandatory attributes successfully verified	PASS	52	00:00:00.9...	11/5/2010 8:50:54.44200...
90	0	PTS Message	PBAP - PSE - Mandatory attributes successfully verified	PASS	55	00:00:00.9...	11/5/2010 8:50:55.36000...
91	0	PTS Message	Service attribute check complete	PASS	32	00:00:00.1...	11/5/2010 8:50:55.51500...
92	0	Test Case Ended	TC_SDAS_BV_03_I	PASS	15	00:00:02.2...	11/5/2010 8:50:57.80000...

Packets that are tagged as captured communications data are forwarded into the HCI protocol layer, and from there fully decoded based on the packet contents. The "tap point" for capturing the messages between the devices is the Host Controller Interface (HCI) between the PTS Application (Host) and the PTS Endpoint Device (Controller).

B...	Frame#	Type	Opcode	Command	Event	Status	Handle	Credits	PBF	Length	Fram...
	79	ACL Data					0x002e	Available Control...	First	18	22
	80	ACL Data					0x002e	Available Host's ...	First	23	27
	81	Event			Number Of Completed Packets		0x002e	Available Host's ...		5	7
	82	ACL Data					0x002e	Available Control...	First	216	220
	84	ACL Data					0x002e	Available Host's ...	First	17	21
	85	Event			Number Of Completed Packets		0x002e	Available Host's ...		5	7
	86	ACL Data					0x002e	Available Control...	First	18	22
	87	ACL Data					0x002e	Available Host's ...	First	23	27
	88	Event			Number Of Completed Packets		0x002e	Available Host's ...		5	7
	89	ACL Data					0x002e	Available Control...	First	146	150
	93	Command	0x0c03	Reset						0	3
	94	Event	0x0c03	Reset	Command Complete	Success				4	6
	95	Command	0x1005	Read_Buffer_Size						0	3
	96	Event	0x1005	Read_Buffer_Size	Command Complete	Success		Initial Host credit...		11	13

The data captured at the HCI “tap point” consists of

- HCI Commands from the PTS to the Endpoint Device;
- HCI Events generated in response to the various commands and asynchronous notifications that may occur during a connection;
- ACL Data containing peer to peer messages based at the L2CAP layer of the Bluetooth stack.

Starting the PTS Protocol Viewer

PTS Protocol Viewer starts automatically with Bluetooth PTS.

Saving and viewing protocol traces

The data captured by the Protocol Viewer may be saved for future reference.

The saved “capture files” may be viewed using the Protocol Viewer in file view mode. A capture file, which has a .CFA extension, may be opened by double clicking on it in Windows Explorer.

Alternatively, the “Capture File Viewer” shortcut found in the “Bluetooth PTS” folder in the Windows start menu may be used to start the Profile Viewer in file view mode. Once started, the “Open Capture File...” item on the “File” menu may be used to open a capture file.

Verdict Determination

Verdict Determination

Test cases have two mechanisms that are used to determine the final result of a test case execution. One method is to simply set the final verdict when a final result can be determined. When the verdict is set in this way, the test case terminates immediately.

It may be however that terminating a test case in the middle of its execution is not desired behavior. For these situations, the “Preliminary verdict” mechanism is used. For this mechanism, “Preliminary verdict” events ([Interesting events](#), “Preliminary Verdict”) are issued at various points during the test:

- Any "Preliminary Verdict" of "FAIL" will result in a final verdict of "FAIL".
- Any "Preliminary Verdict" of "INCONclusive" will result in a final verdict of "INCONclusive", unless a "Preliminary verdict" of "FAIL" was also issued during the test run.
- A final verdict of "PASS" will only occur when all "Preliminary verdicts" are "PASS".

As an example, a test case may be broken into three segments: startup, the real work of the test, and termination. If the startup segment executes successfully, it may be necessary to execute the termination segment regardless of whether or not the real work of the test succeeds. It might be that the results of the segments are:

Segment	Preliminary verdict
Startup	(PASS)
Real work	(FAIL)
Termination	(PASS)

In this case, it may be necessary to execute the termination segment in order to cleanly disconnect from the IUT. The termination segment succeeded, but the overall result of the test was a verdict of "FAIL", because the middle segment set a "Preliminary verdict" of "FAIL".

A final verdict of "NONE" is issued to a test case when its execution is canceled by users or terminated before a "Preliminary Verdict" is given.

Verdict Determination

Test cases have two mechanisms that are used to determine the final result of a test case execution. One method is to simply set the final verdict when a final result can be determined. When the verdict is set in this way, the test case terminates immediately.

It may be however that terminating a test case in the middle of its execution is not desired behavior. For these situations, the "Preliminary verdict" mechanism is used. For this mechanism, "Preliminary verdict" events ([Interesting events](#), "Preliminary Verdict") are issued at various points during the test:

- Any "Preliminary Verdict" of "FAIL" will result in a final verdict of "FAIL".
- Any "Preliminary Verdict" of "INCONclusive" will result in a final verdict of "INCONclusive", unless a "Preliminary verdict" of "FAIL" was also issued during the test run.
- A final verdict of "PASS" will only occur when all "Preliminary verdicts" are "PASS".

As an example, a test case may be broken into three segments: startup, the real work of the test, and termination. If the startup segment executes successfully, it may be necessary to execute the termination segment regardless of whether or not the real work of the test succeeds. It might be that the results of the segments are:

Segment	Preliminary verdict
Startup	(PASS)
Real work	(FAIL)
Termination	(PASS)

In this case, it may be necessary to execute the termination segment in order to cleanly disconnect from the IUT. The termination segment succeeded, but the overall result of the test was a verdict of "FAIL", because the middle segment set a "Preliminary verdict" of "FAIL".

A final verdict of "NONE" is issued to a test case when its execution is canceled by users or terminated before a "Preliminary Verdict" is given.

Index

A

Abort, Retry	65, 86
Aborting	44
test case execution	44
About ... 10, 13, 44, 46, 58, 65, 70, 77, 79, 86, 101, 108, 109, 147	
Abstract Test Suite	35, 62
accompanying	62
Accompanying	62
Abstract Test Suite	62
ACL Data	155
ACL Data containing	155
Activating	68
custom Implicit Send DLL	68
Active project	131
Add Product	109, 121
Adding	7, 133, 137, 141
project	7
Test Case	141
Test Cases	133
Test Script	137
AddRef	86
Address	44, 58, 77, 79, 86
ImplicitSendStyle	58
Adds communication	46
After	17
AG	30
Air Sniffer	44
All Groups	17
Allows ... 7, 35, 44, 53, 58, 65, 70, 73, 74, 86, 121, 131, 149, 155	
Client Application	86
Already be	1
Already be selected	1
Also be used at	1
Also be used at this time	7
Also issued	158, 161
Always	46, 62, 79, 149
Always return	79
API	53, 72, 73, 75, 86, 101, 131
API Error Codes	75, 101
Application Programming Interface	53, 73, 131
Application Settings	46, 152, 157
Applications .. 17, 46, 53, 73, 86, 99, 107, 109, 131, 143, 152, 155	
existing	86
ASCII	75
ATS	35, 62
Consult	62
Audio Gateway	30
Automatic dismissal	70
Implicit Send requests	70
Automating	53, 73, 86, 131
PTS	53, 73, 131

Automating Test Execution	50
Automation	55, 56, 57, 65, 70, 99
Automation test platforms	55, 57, 65, 70
Autotest	74
AVRCP	10

B

Babysit	55
Back	13, 41, 56, 109, 121
Basic Information	57
Basic Rate	17
BD_ADDR	1, 9, 44
device is	1
BDADDR	75, 98
IUT	75
receives	98
Before	17
Beginning	62
strMmiText	62
Beta test suites	13
BITSTRING	84
Bluetooth 1, 9, 10, 17, 30, 35, 44, 53, 56, 62, 71, 73, 75, 98, 99, 101, 107, 143, 152, 155, 157	
controlling	73
describe	107
details	56
layer	155
lifecycle	107
Bluetooth application	17
Bluetooth BR	143
Bluetooth Controller	143
Bluetooth Device ... 1, 9, 10, 35, 44, 73, 75, 98, 101, 107	
contains	35
Bluetooth Device Address	1, 9, 35, 44, 75, 98, 101
contains	75
determine	101
lists	44
retrieves	98
Bluetooth Host	143
Bluetooth Low Energy	17, 143, 152
Bluetooth PTS	1, 62, 71, 73, 99, 101, 157
Bluetooth SIG	1, 62, 71, 73, 99, 101, 107
Bluetooth Special Interest Group	53
Bool	57, 58, 79, 86
BOOL bValue	84
BOOL value	79
Bool WINAPI InitImplicitSend	58
BOOLEAN	35, 84
BR	17, 152
compared	152
Build	71, 73, 75, 77, 98, 109
Client Application	75
Button 1, 13, 17, 30, 35, 44, 45, 55, 65, 68, 70, 86, 109, 121, 129, 135	
PTS toolbar	45

-
- Button will open 1, 109
 - Buttons Displayed 65, 86
 - BV 9
 - BValue 84
 - C**
 - C/C 65
 - Calling 74, 77, 79, 101
 - CoCreateInstance 74
 - GetProjectName 77
 - GetPTSBluetoothAddress 101
 - GetTestCaseDescription 79
 - GetTestCaseName 79
 - Implicit Send 101
 - UpdatePics 101
 - Windows API 74
 - Calling UpdateICS 101
 - Cancel 65, 86, 121, 158, 161
 - Cancel button 86
 - include 86
 - pressed 86
 - Capture File Viewer 157
 - Case .. 17, 27, 30, 40, 41, 44, 46, 50, 58, 65, 75, 77, 79, 84, 99, 101, 109, 126, 129, 143, 145, 147, 153, 158, 161
 - PTS Development Team 41
 - Case shown 126
 - Cause .. 17, 30, 44, 46, 58, 62, 68, 70, 74, 109, 145, 152
 - Client Application 74
 - Windows Explorer window 145
 - Center 146
 - PTS 146
 - CFA 157
 - Chain 56
 - MMI PTC 56
 - Change .. 1, 7, 9, 17, 30, 35, 41, 46, 58, 67, 68, 86, 108, 109, 126, 137, 139, 141, 149, 152, 153
 - order 139
 - PICS 7, 30
 - Check 17, 27, 30, 46, 62, 68, 71, 126
 - checkbox 30
 - Checkbox 13, 17, 30, 109, 124, 126, 127, 153
 - check 30
 - item 126
 - Unchecking 109
 - Checkboxes 17, 127
 - corresponding 127
 - Checkmark 13, 70, 109, 127
 - place 109, 127
 - removing 70
 - Checkmarks 109
 - Choose 13, 109, 133
 - Test Case 133
 - Class 1, 57, 58
 - Device 1
 - Class Of Device 1
 - Clear Search Results 1
 - Clicking 30, 35, 79, 127
 - IXIT 35
 - PICS 30
 - PIXIT 35
 - Removed Checked Items button 127
 - Test Case 79
 - Client Application application 86
 - log 86
 - Client Applications 73, 74, 75, 86
 - allows 86
 - build 75
 - cause 74
 - existing 86
 - MMIs 86
 - prompts 86
 - ClientShowBdAddress 98
 - Close 58, 157
 - CO_E_SERVER_EXEC_FAILURE 101
 - CoCreateInstance 74, 101
 - call 74
 - Codename 1
 - ColInitialize 74, 75
 - ColInitializeEx 74, 75
 - COM 73, 74, 86, 99, 101
 - create 86
 - COM during 73
 - COM Server 74, 99, 101
 - COM Server Mode 74, 99, 101
 - Combining 17
 - Filters 17
 - Comes 75, 107
 - Compared 152
 - BR 152
 - Component Object Model 73, 74, 86
 - Component Object Module 101
 - Connected 17, 55, 71, 74, 79, 101
 - PTS Control API 79
 - Consolidate 107
 - used 107
 - Consult 62, 86
 - ATS 62
 - Consult implicit_send_log 62
 - Containing 1, 30, 35, 75, 77, 79, 109
 - Bluetooth Device 35
 - Bluetooth Device Address 75
 - dropdown list of 109
 - PICS 1
 - Project 77
 - PTS 30
 - Test Case 79
 - Contents .. 41, 65, 77, 79, 84, 86, 99, 108, 143, 146, 155
 - Output window 143
 - report 108
 - strMmiText 65
 - Control-click 13
 - Controlling 73
 - Bluetooth 73
 - Copied 13, 65, 86, 109, 141, 143
 - Windows 109, 143

Copy Highlighted.....	109
Corner.....	1, 86, 143, 145
PTS.....	143, 145
PTS User Interface.....	86
Correlate.....	155
Protocol Viewer.....	155
Corresponding.....	79, 127
checkboxes.....	127
Test Purpose.....	79
Coupled.....	157
PTS Protocol Viewer application.....	157
Cpp.....	58, 71
CreateWorkspace.....	75, 99, 101
demonstrate.....	99
Creating.....	1, 75, 86, 99, 101, 109, 132
COM.....	86
initial Test Script.....	132
new workspace.....	1
PTS Report.....	109
Test Script.....	99
time.....	109
used.....	86, 99, 109
Workspace.....	75, 99, 101
Current Link Key.....	44, 45
Deleting.....	45
Currently associated.....	109
Currently, StopTestCase.....	101
Custom.....	50, 57, 58, 62, 67, 68, 70, 71, 72, 73, 86, 99, 152
Custom DLLs.....	58, 67, 70, 72
Custom Implicit Send DLL.....	50, 57, 62, 67, 68, 70, 71, 72, 86
Activating.....	68
Cut.....	139, 141

D

Data are.....	124, 155
Data is.....	65, 109
Data type.....	35, 57, 73, 84, 101
item that.....	35
Deallocated.....	75
Debug.....	41, 46, 67, 75, 145, 152
use.....	75
Debug Logs.....	41, 46, 145, 152
Default Settings.....	17
Delete.....	8, 130
Delete Link Key.....	45
Delete Product.....	121
Deleting.....	8, 9, 45, 58, 109, 121, 124, 127, 130, 137
current Link Key.....	45
test case results.....	130
workspace.....	8
Deleting Device Descriptions.....	121
Deletion.....	126, 127, 129, 130
Demonstrate.....	9, 98, 99, 107
CreateWorkspace.....	99

Describe.....	1, 9, 30, 35, 40, 44, 53, 57, 58, 62, 65, 73, 75, 86, 99, 107, 109, 131, 141, 145
Bluetooth.....	107
Describes this feature.....	53, 73
Description.....	10, 30, 35, 79, 86, 108, 109, 121, 143
Selection.....	10
Description ending.....	30
Design.....	1, 53, 72
Detail that is present.....	46
Details.....	46, 50, 56, 58, 62, 108, 109, 121, 126, 143, 152, 155
Bluetooth.....	56
Details regarding.....	58
Determine.....	10, 30, 35, 46, 50, 58, 62, 86, 98, 101, 109, 149, 152, 158, 161
Bluetooth Device Address.....	101
Develop.....	57, 71, 72, 73, 86, 143
used.....	73
Development checkpoints.....	108
Development Teams.....	46, 152
Device.....	1, 7, 9, 17, 27, 30, 35, 44, 45, 53, 55, 65, 70, 75, 101, 107, 108, 109, 121, 143, 155
Class.....	1
of testing.....	107
Device description.....	108, 109, 121
Device display.....	65
Device is.....	1, 70, 108, 155
BD_ADDR.....	1
IXIT item.....	1
Device is tested.....	108
Device Name.....	1
Dialog will appear.....	7, 109
Dialog will appear allowing.....	7
Dialog will appear asking.....	109
Dialog will display.....	124
list of.....	124
Direct.....	107, 109
used.....	107
Disable.....	30, 46, 58, 70, 71, 79, 101, 149, 153
Implicit Send.....	71
Disable this functionality.....	46
Disconnects.....	86, 158, 161
Implicit Send.....	86
Displayed in.....	17, 27, 40, 86, 145, 147, 149, 153, 155
log.....	153
Displaying.....	40, 149
PICS.....	149
purpose.....	40
Distinguish.....	62, 109
used.....	109
DLL.....	57, 58, 67, 68, 70, 71, 72, 73, 101
find.....	57
lead.....	58
load.....	57
locate.....	68
unloading.....	58
DLL_PROCESS_DETACH.....	58

-
- DllMain*.....58
 - use*.....58
 - Documentation*.....35, 86, 99, 109
 - Dropdown list*.....121
 - Using*.....121
 - Dropdown list of*.....109
 - contains*.....109
 - DUN*.....13
 - During*.....53, 58, 73, 86, 98, 131
 - execution of*.....53, 73, 131
 - ImplicitStartTestCase*.....58
 - PTS*.....98
 - Test Case*.....86
 - DWORD*.....77, 86, 98
 - DWORD responseSize*.....86
 - E**
 - E_NOINTERFACE*.....101
 - Each test*.....9, 10, 27, 35, 40, 58, 62, 72, 99, 108, 109, 126, 129, 145
 - Each test case*.....27, 40, 58, 99, 109, 126, 129, 145
 - Each test case shown*.....126
 - Edit Product*.....109, 121
 - Edit Tester*.....109
 - Edited before*.....1
 - Editing*.....1, 30, 35, 124, 137
 - PICS*.....1
 - project PICS*.....30
 - project PIXIT settings*.....35
 - Test History*.....124
 - Test Script*.....137
 - EDR*.....17, 143, 152
 - Enable* . 30, 44, 46, 50, 53, 62, 67, 71, 73, 79, 107, 135, 153
 - used*.....71
 - Encrypted Verdict*.....108, 149
 - Endpoint*.....44, 98, 101, 155
 - Endpoint Device*.....98, 101, 155
 - PTS*.....155
 - Endpoint information*.....44
 - Enhanced Data Rate*.....17
 - Enter* ... 1, 17, 35, 44, 53, 58, 65, 68, 101, 109, 121, 135
 - PIN Code*.....58
 - Entry titled*.....62
 - Error*.....65, 75, 77, 79, 84, 86, 98, 101
 - Error occurs*.....65, 101
 - ETS*.....9, 58
 - Event that is logged when a Test Case*.....86
 - Events* . 86, 143, 146, 147, 149, 152, 153, 155, 158, 161
 - information displayed*.....147
 - Interesting*.....149
 - Selecting*.....152
 - Every MMI*.....86
 - Excel*.....99
 - Exe*.....74, 75, 98, 99, 101
 - Executable Test Suite*.....9, 58, 62, 101
 - identifies*.....62
 - Executable Test Suite DLLs*.....101
 - Executables*.....9, 58, 62, 67, 99, 101, 108, 147
 - Executing*.....41, 62, 86, 135
 - single test case*.....41
 - test case*.....62, 86
 - Test Script*.....135
 - Execution*30, 41, 46, 53, 58, 65, 70, 73, 74, 79, 86, 101, 108, 109, 124, 129, 131, 135, 139, 141, 143, 145, 146, 147, 149, 152, 153, 158, 161
 - order*.....139
 - Execution log*.....46, 79, 86, 143, 146, 147, 149, 152
 - Format*.....147
 - Execution of*...41, 53, 58, 70, 73, 74, 86, 101, 109, 131, 135, 143, 147, 149
 - during*.....53, 73, 131
 - Existing*.....75, 86, 99, 131
 - Applications*.....86
 - Client Applications*.....86
 - PTS Workspace*.....131
 - Workspace*.....75, 99
 - Exit*.....8, 58, 99, 101, 157
 - PTS*.....157
 - PTS application*.....8
 - Expected*.....9, 53, 56, 58, 65, 86
 - Expected Outcome*.....9, 53
 - Expects one*.....58
 - Expects one of*.....58
 - Explorer*.....145
 - Export*.....1, 75, 99
 - Extended Automating*.....53, 131
 - F**
 - FAIL*.....27, 86, 158, 161
 - FAILED*.....129
 - False*.....30, 58, 79
 - setting pbResponselsPresent*.....86
 - FALSE if*.....86
 - File name*.....5, 58, 109
 - Filters*.....1, 10, 13, 17
 - Combining*.....17
 - Final verdict*....46, 79, 86, 143, 145, 149, 152, 158, 161
 - Output window along with*.....143
 - Find*.....50, 57, 62
 - DLL*.....57
 - Finish*.....1, 13, 41, 109
 - Format*.....9, 62, 75, 99, 107, 108, 147
 - execution log*.....147
 - From*.....1, 5, 8, 9, 13, 17, 30, 35, 41, 46, 50, 56, 57, 58, 65, 68, 70, 72, 73, 74, 75, 77, 79, 86, 98, 99, 101, 109, 121, 124, 127, 131, 133, 135, 137, 139, 146, 149, 153, 155, 158, 161
 - pszWorkspaceName*.....75
 - From calling UpdateICS*.....101
 - From the*...1, 5, 8, 13, 17, 30, 35, 41, 50, 56, 57, 65, 68, 70, 72, 75, 79, 86, 99, 109, 121, 124, 127, 131, 133, 137, 139, 149, 153, 155, 158, 161

From the Implementation Under Test	86
From the pszWorkspaceName.....	75
From the server.....	149
FTP.....	13, 62
Full.....	10, 17, 41, 46, 75, 99, 101, 152
Fully Automated Operation.....	53, 73, 131

G

GATT.....	17
General.....	9, 46, 50, 75, 98, 101, 149, 153
value represents.....	101
General Application Settings	46
General information functions	98
General interest.....	149
General Usage.....	75
Generate	65, 108, 109, 124, 155
Generate Report	109
Generate report with text logging	109
Generating Detailed Log.....	109
GetProcAddress.....	72
GetProjectCount	77
GetProjectName	77, 101
calling.....	77
value	101
GetProjectVersion.....	77, 101
GetPTSBluetoothAddress	98, 101
call.....	101
GetPTSVersion	98, 101
GetTestCaseCount	79
GetTestCaseDescription.....	79, 101
calling.....	79
GetTestCaseName	79, 101
calling.....	79
value	101
Good idea.....	1, 67, 121
Good idea to execute.....	1
Group.....	9, 13, 17, 27, 44, 46, 53, 73, 131

H

Handling	58
Implicit Send	58
Handsfree.....	30
Handsfree Profile	30
HandsFree Profile Test Specification.....	109
HCI	155
HCI Commands.....	155
HCI Events	155
HCI?HCI_READ_LOCAL_VERSION_INFORMATION_	
COMPLETE_EVENT	86
HCI_OK.....	86
HciRevision	86
HciVersion	86
Help	46, 62, 109

HFP.....	30
HFP15.....	10
High Speed	17
Highlight.....	108, 126, 137, 139, 143
Test Case.....	139
History	109, 124
Host Controller Interface	155
HRESULT	75, 77, 79, 84, 86, 98
return	75
HRESULT CreateWorkspace.....	75
HRESULT GetProjectCount.....	77
HRESULT GetProjectName.....	77
HRESULT GetProjectVersion	77
HRESULT GetPTSBluetoothAddress.....	98
HRESULT GetPTSVersion	98
HRESULT GetTestCaseCount.....	79
HRESULT GetTestCaseDescription.....	79
HRESULT GetTestCaseName.....	79
HRESULT IsActiveTestCase	79
HRESULT Log	86
HRESULT OnImplicitSend	86
HRESULT OnSend.....	86
HRESULT OpenWorkspace.....	75
HRESULT RegisterImplicitSendCallback.....	86
HRESULT RegisterImplicitSendCallbackEx.....	86
HRESULT RunTestCase.....	79
HRESULT SetControlClientLoggercallback	86
HRESULT StopTestCase.....	79
HRESULT UnregisterImplicitSendCallback.....	86
HRESULT UnregisterImplicitSendCallbackEx	86
HRESULT UpdateICS	84
HRESULT UpdateIXITParam.....	84
HRESULT UpdatePics	84
HRESULT UpdatePixitParam	84
HS	17
Hybrid environments.....	72

I

IA5STRING	35, 84
ICS1, 7, 9, 30, 35, 75, 84, 101, 108, 109, 131, 132, 147,	
149	
ICS from TPG	1, 7, 30
ICS Proforma	30
ICS used by	1
Identifies.....	9, 10, 62, 72, 86, 149, 155
Executable Test Suite.....	62
MMI.....	86
Identifies the.....	58, 62, 86
Identifies the executing.....	62
IEEE 802.11.....	17

-
- If 1, 7, 10, 13, 17, 44, 46, 53, 57, 58, 62, 65, 70, 72, 74, 75, 77, 79, 84, 86, 98, 99, 101, 107, 109, 121, 124, 135, 141, 145, 146, 149, 153, 158, 161
 - Test Case 135
 - Ignore 65, 75, 86, 99
 - Implement 35, 56, 58, 67, 73, 79, 86, 101
 - used 56
 - Implementation Under Test. 1, 9, 30, 41, 53, 56, 75, 86, 143, 155
 - PTS 86
 - Selecting 1
 - Implemented 56
 - Implicit 50, 56, 57, 58, 62, 65, 67, 68, 70, 71, 72, 86, 101
 - Implicit Send 50, 56, 57, 58, 62, 65, 67, 68, 70, 71, 72, 86, 101
 - call 101
 - disable 71
 - disconnects 86
 - handling 58
 - PTS 58
 - use 86
 - Implicit Send API 58
 - Implicit Send DLL. 50, 56, 57, 58, 62, 65, 67, 68, 70, 71, 72, 86
 - Implicit Send DLL creates 62
 - log 62
 - Implicit Send DLL display 62
 - tag 62
 - Implicit Send functions 58
 - Implicit Send handling 72
 - Implicit Send requests 58, 62, 70
 - Automatic dismissal 70
 - Implicit_send 71
 - Implicit_send_log 62
 - ImplicitSend 58, 62, 71
 - ImplicitSendPinCode 57, 58, 70
 - ImplicitSendStyle 57, 58, 62, 65, 70, 71, 72, 86
 - address 58
 - ImplicitStartTestCase 57, 58
 - during 58
 - ImplicitTestCaseFinished 57, 58
 - Import PICS 1, 30
 - Import PICS from TPG 1, 7
 - Import Test Plan 1
 - Imported 1, 7, 30, 108
 - PICS 1
 - PTS 1
 - Include. 10, 17, 46, 50, 62, 71, 75, 86, 98, 99, 107, 108, 109, 129, 131, 147, 152
 - Cancel button 86
 - Included making 71
 - Including communicating 98
 - Inclusion 13, 108, 109
 - test suites 13
 - INCONCLUSIVE 27, 109, 158, 161
 - Indicates 10, 17, 27, 30, 35, 44, 65, 71, 86, 101, 108, 109, 149
 - PTS Control API 86
 - used 27, 86
 - Individual test case results 127
 - Selecting 127
 - Information. 1, 10, 13, 41, 44, 46, 50, 58, 65, 67, 70, 73, 75, 77, 79, 86, 107, 108, 109, 121, 143, 146, 147, 149, 152, 153, 155
 - of the test case 46
 - Information displayed 46, 86, 147, 155
 - event 147
 - Information during 109
 - Information that has been encrypted 108
 - Information used to 46
 - Ini 62
 - Initial Test Script 132
 - Creating 132
 - Initializations 58, 149
 - Initialized 74, 77, 79, 101
 - NULL 77, 79
 - InitImplicitSend 57, 58, 72
 - Inquiry Request 1
 - Insert 7, 9
 - Instruct 65
 - used 65
 - INTEGER 35, 84
 - Interesting 149
 - events 149
 - Internet 109
 - Internet Explorer 109
 - Introduction 107, 143
 - IOPT 99
 - IP 71
 - IProject 77
 - IProject value 77
 - IPTSControl COM 74
 - IPTSControlClientLogger 86, 101
 - IPTSImplicitSendCallback 86, 101
 - IPTSImplicitSendCallbackEx 86
 - refer 86
 - use 86
 - IsActiveTestCase 79, 101
 - Item 1, 7, 9, 10, 13, 17, 27, 30, 35, 46, 50, 53, 55, 58, 65, 71, 75, 84, 86, 101, 108, 109, 124, 126, 127, 135, 137, 141, 146, 149, 152, 157
 - checkbox 126
 - PTS 1
 - when selecting 126
 - Item applies 30
 - Item contains 35
 - Item that 10, 13, 35
 - data type 35
 - Item will 30, 109, 124, 126, 135
 - Items declared 75

<i>Items should always</i>	30
<i>Items that</i>	10, 30, 124, 127
<i>ITestCase</i>	79
<i>ITestCase value</i>	79
<i>Its own</i>	58, 149
<i>IUT ...</i> 1, 9, 27, 30, 41, 44, 45, 53, 56, 62, 65, 70, 75, 86, 149, 158, 161	
<i>BDADDR</i>	75
<i>PTS</i>	65
<i>Using</i>	70
<i>IUT Device Address</i>	1
<i>IUT to PTS</i>	149
<i>IXIT ...</i> 1, 9, 30, 35, 71, 84, 101, 108, 109, 131, 147, 149	
<i>clicking</i>	35
<i>Opening</i>	35
<i>updating</i>	101
<i>IXIT changes that</i>	101
<i>IXIT data</i>	84, 109
<i>IXIT documentation</i>	35
<i>IXIT items</i>	1, 9, 35, 84, 101
<i>device is</i>	1
<i>IXIT settings</i>	35, 108, 109, 131
<i>IXIT table</i>	35
<i>project</i>	35
<i>IXIT Tool Window</i>	30, 35, 132
<i>IXIT value</i>	71, 101, 147, 149

L

<i>L</i> 77, 79	
<i>L2CAP</i>	17, 155
<i>Layer</i>	17, 56, 149, 155
<i>Bluetooth</i>	155
<i>LE17, 143, 152</i>	
<i>Lead</i>	58, 86, 153
<i>DLL</i>	58
<i>Legacy</i>	5
<i>selecting</i>	5
<i>Legal Characters For The New Value</i>	84
<i>Less Operation</i>	53, 86, 131
<i>Lifecycle</i>	107
<i>Bluetooth</i>	107
<i>Lines</i>	62, 65, 74, 86, 101, 145, 147, 149
<i>PTS</i>	62
<i>Link Key</i>	44
<i>List of ...</i> 5, 10, 17, 27, 35, 58, 65, 75, 77, 79, 84, 86, 98, 108, 124, 137	
<i>dialog will display</i>	124
<i>Listing</i>	44, 108
<i>Bluetooth Device Address</i>	44
<i>PICS</i>	108
<i>LI</i> 57	
<i>LmpSubversion</i>	86
<i>LmpVersion</i>	86
<i>Load</i>	57, 58, 72, 75
<i>DLL</i>	57

<i>Locate ...</i> 1, 8, 10, 13, 17, 41, 44, 57, 68, 73, 75, 98, 143	
<i>DLL</i>	68
<i>Location</i>	1, 57, 71, 75, 108, 109
<i>Log</i> 41, 46, 50, 62, 79, 86, 101, 108, 109, 126, 143, 145, 146, 147, 149, 152, 153	
<i>Client Application application</i>	86
<i>displayed in</i>	153
<i>Implicit Send DLL creates</i>	62
<i>Log containing</i>	62
<i>Logfile</i>	145
<i>Logfiles</i>	145
<i>Logging support</i>	143
<i>LogType</i>	86
<i>values</i>	86
<i>LogType Value</i>	86
<i>Long time</i>	109
<i>take quite</i>	109
<i>Low</i>	17, 143, 152
<i>LPCWSTR pszDescription</i>	86
<i>LPCWSTR pszEntryName</i>	84
<i>LPCWSTR pszMessage</i>	86
<i>LPCWSTR pszNewParamValue</i>	84
<i>LPCWSTR pszParamName</i>	84
<i>LPCWSTR pszPathOfPtsFile</i>	75
<i>LPCWSTR pszPathOfWorkspace</i>	75
<i>LPCWSTR pszProjectName</i>	77, 79, 84, 86
<i>LPCWSTR pszTestCase</i>	79, 86
<i>LPCWSTR pszWorkspaceName</i>	75
<i>LPCWSTR pszWorkspacePath</i>	75
<i>LPCWSTR szLogType</i>	86
<i>LPCWSTR szTime</i>	86
<i>LPWSTR</i>	77, 79, 86
<i>LPWSTR pszProjectName</i>	77
<i>LPWSTR pszResponse</i>	86
<i>LPWSTR pszTestCaseDesc</i>	79
<i>LPWSTR pszTestCaseName</i>	79

M

<i>Main Test Component</i>	56
<i>using</i>	56
<i>Major Device Class</i>	1
<i>Malloc</i>	58
<i>ManufacturerName</i>	86
<i>Many Bluetooth SIG</i>	107
<i>Many DLLs</i>	72
<i>Menu contains</i>	109
<i>Menu may</i>	157
<i>Message tags</i>	58, 62, 65, 72
<i>Message Type</i>	65, 86
<i>Microsoft</i>	58, 67, 73
<i>Microsoft Visual</i>	67, 73
<i>Microsoft's Component Object Model</i>	73

-
- Minimal 46, 152
 - MMI 56, 58, 65, 86
 - Client Application 86
 - identifies 86
 - presents 86
 - style 86
 - MMI Handler 56
 - MMI PTC 56
 - chain 56
 - MMI PTC handles 56
 - MMI style defines 86
 - MMI styles 65, 86
 - MMI_Style_Abort_Retry1 65, 86
 - MMI_Style_Edit1 58, 65, 86
 - MMI_Style_Edit2 58, 65, 86
 - MMI_Style_Ok 65, 86
 - MMI_Style_Ok_Cancel1 65, 86
 - MMI_Style_Ok_Cancel2 65, 70, 86
 - MMI_Style_Yes_No_Cancel1 65, 86
 - MMI_Style_Yes_No1 65, 86
 - MmiStyle 58, 65
 - MmiStyle name 65
 - MmiStyle value 65
 - More 1, 9, 10, 13, 17, 46, 50, 58, 67, 70, 72, 79, 86, 99, 108, 109, 121, 129, 131, 143, 146, 149
 - More complicated 72
 - Most recent 109
 - Most Test Cases 86
 - Ms 86
 - MTC 56
 - MultiByteToWideChar 75
 - My Workspaces 1, 62, 99
 - MyLaptop 65
 - MyPda 65
 - MyPhone 65
 - N**
 - N; ASCII 65
 - Name. 1, 5, 9, 10, 13, 17, 27, 30, 35, 41, 50, 57, 58, 62, 65, 71, 72, 75, 77, 79, 84, 86, 99, 101, 108, 109, 121, 131, 132, 145, 147, 149, 153
 - PICS 30
 - PIXIT 35
 - Project 99
 - PTS 109
 - subfolder 75
 - Test Cases 131
 - Test Script 101
 - Workspace 75
 - Workspace file 75
 - Name corresponding 9
 - Name of 1, 13, 27, 30, 35, 41, 58, 75, 77, 79, 84, 86, 99, 101, 109, 131, 145, 147, 149
 - with 75
 - Name of the workspace 1, 75
 - Name>SampleTest 99
 - Needed. 1, 27, 35, 41, 44, 46, 50, 53, 55, 56, 57, 58, 62, 65, 67, 70, 73, 74, 75, 86, 99, 101, 108, 109, 121, 124, 126, 129, 131, 133, 152
 - Visual Studio 67
 - New 1, 10, 58, 62, 65, 75, 84, 86, 99, 101, 109, 121, 131
 - New Client Applications 86
 - New workspace 1, 75, 99
 - Creating 1
 - Newline 65
 - Next 1, 17, 27, 44, 68, 108, 109, 121, 147
 - NmyLaptop 65
 - NmyPda 65
 - NmyPhone 65
 - No buttons 86
 - NONE 158, 161
 - NOT 58, 86
 - NOT call 58
 - Notepad 99
 - NUL 75, 86
 - room for 86
 - NULL 58, 65, 77, 79, 86
 - initialized 77, 79
 - return 65
 - returns 86
 - Number 9, 10, 17, 30, 41, 56, 62, 65, 72, 77, 79, 86, 98, 99, 107, 108, 109, 143, 146, 147, 152
 - Parallel Test Components 56
 - PTS 98
 - Test Cases 99
 - Test cases produce 41
 - O**
 - OBEX 149
 - shows 149
 - OBEX_CONNECT_REQ 149
 - Object 53, 58, 62, 67, 74, 75, 86
 - PTS Control API 86
 - Object Push 53, 62
 - Object Push Profile 62
 - OCTETSTRING 35, 84, 101
 - returned 101
 - Of 1, 5, 9, 10, 13, 17, 27, 30, 35, 40, 41, 44, 46, 50, 53, 55, 56, 57, 58, 62, 65, 67, 68, 70, 71, 72, 73, 74, 75, 77, 79, 84, 86, 98, 99, 101, 107, 108, 109, 121, 124, 126, 127, 129, 131, 132, 133, 135, 141, 143, 145, 146, 147, 149, 152, 153, 155, 158, 161
 - pszReponse 86
 - pszResponse 86
 - strMmiText 62, 65
 - test case 27, 50, 56, 79, 86, 108, 109, 143, 145, 146, 149, 152, 153
 - test case executions 86, 109, 143, 149
 - Test Case View 50, 153
 - Test Suite Selector will display 10
 - Of creating 109
 - report 109

Of gathering	41
Of preparing	1
Of range	101
Of testing	9, 107
device	107
Of the1, 9, 10, 13, 17, 27, 30, 35, 40, 41, 44, 46, 50, 53, 56, 57, 58, 62, 65, 67, 68, 70, 73, 74, 75, 77, 79, 84, 86, 98, 99, 101, 107, 108, 109, 121, 124, 127, 131, 132, 133, 141, 143, 145, 146, 147, 149, 152, 153, 155, 158, 161	
Of the.....	50
Of the.....	153
Of the corresponding	79
Of the following	53, 73, 131
Of the item that.....	35
Of the IXIT items	35
Of the pszReponse	86
Of the pszResponse	86
Of the resulting.....	41
Of the string.....	65, 77, 79
Of the strMmiText	62, 65
Of the strMmiText string	62
Of the test case.....	27, 30, 46, 50, 56, 79, 86, 108, 109, 124, 143, 145, 146, 149, 152, 153
information	46
Of the Test Case execution	86, 149
Of the test case executions	86, 109, 124, 143
of the test case executions	124
Of the test case runs for	109
Of the test case to.....	46
of the test case to.....	46
Of the test cases available	30
of the test cases available.....	30
Of the test cases in	30
of the test cases in	30
Of the Test Suite Selector will display.....	10
Of viewing.....	126
OK.....	58, 65, 68, 86, 109, 121
presses	65
OK button	68, 86
press	68
Older test results.....	129
Selecting	129
OnImplicitSend.....	86
OnSend	86, 101
Open Capture File.....	157
Open File.....	58
Open Workspace	5, 53, 73, 75, 77, 109, 131, 145
Opening.....	30, 35
IXIT	35
PICS.....	30
PIXIT.....	35
Opening/Creating.....	75
Workspace.....	75

OpenWorkspace.....	75, 101
Operation1, 46, 53, 55, 58, 62, 65, 70, 73, 86, 121, 124, 130, 131	
Operation which may.....	1
Operator.....	41, 53, 55, 56, 58, 62, 65, 70, 73, 86, 131
OPHBV03	62
OPHBV07	62
OPP.....	10, 62
OPP Profile	62
Options.....	46, 50, 62
Or other	53, 55, 149
Order..9, 17, 27, 35, 44, 53, 57, 58, 62, 73, 74, 99, 107, 109, 124, 131, 133, 135, 137, 139, 158, 161	
Changing	139
execution	139
PTS.....	57
Order of.....	139, 141
Order of execution	139, 141
Org	1
Output Window	86, 143, 147, 149
contents.....	143
Output window along with	143
final verdict	143

P

Page allows.....	152
selection	152
Page of.....	152, 153, 157
PTS application	157
Parallel Test Components.....	56
number	56
Parameter Name	35
Part.....	1, 13, 30, 53, 56, 62, 73, 86, 99, 108, 149
PTS.....	73, 99
PASS.....	27, 30, 86, 109, 129, 158, 161
Passkey Entry.....	65
PBAP.....	10, 149
PbIsActive	79
PbResponselsPresent.....	86
setting.....	86
PCallback.....	86
PcProjects.....	77, 79
PcTestCases.....	79
PDU.....	86
PersistentText.....	58
PersistentText.cpp/h.....	58
Phone Book Access Profile	149
PTS.....	149
PICS.....	1, 7, 9, 30, 35, 75, 84, 101, 108, 109, 131, 132, 147, 149
change.....	7, 30
clicking.....	30
containing.....	1
display	149
edit	1

importing.....	1	Profile.....	1, 7, 9, 10, 13, 17, 27, 30, 35, 40, 46, 53, 56, 72, 73, 75, 107, 108, 109, 131, 132, 143, 157
listing.....	108	project represents.....	9
names.....	30	Profile & Protocol.....	17
Opening.....	30	Profile Implementation Conformance Statement.....	9
set.....	75	Profile Implementation Extra Information.....	9
updating.....	101	Testing.....	9
Using.....	30	Profile Name.....	132
PICS file.....	75, 101	Profile Tuning Suite.....	9, 46, 53, 73, 107, 109, 131, 143
PICS items.....	30, 84, 101	Use.....	109
value.....	84	Profile Viewer.....	157
PICS used.....	1	Program Files.....	1, 62, 71, 73, 99, 101
PIN.....	58	Project PICS.....	30
PIN Code.....	58	Editing.....	30
enter.....	58	Project PIXIT settings.....	35
PIXIT.....	9, 35, 71, 84, 101, 108, 109, 131, 147, 149	Editing.....	35
clicking.....	35	Project represents.....	9
names.....	35	profile.....	9
Opening.....	35	Project Settings.....	46, 50, 68, 70, 153
updating.....	101	returning.....	70
Using.....	35	Project Settings dialog.....	68
PIXIT data.....	84	Project Settings menu.....	68
PIXIT item.....	1, 35, 84, 101	Projects 1, 7, 8, 9, 27, 30, 35, 46, 50, 53, 67, 68, 70, 71, 73, 75, 77, 79, 84, 86, 99, 101, 108, 109, 124, 131, 132, 145, 146, 153	
PIXIT Tool Window.....	30, 35, 132	Adding.....	7
Place.....	13, 41, 58, 68, 70, 86, 109, 127, 139, 143	containing.....	77
checkmark.....	109, 127	IXIT table.....	35
PLogger.....	86	name.....	99
PProjVersion.....	77	Removing.....	9
PpszProjectName.....	77	used to represent.....	8
PpszTestCaseDesc.....	79	Prompts.....	46, 53, 55, 65, 73, 74, 86, 101, 131
PpszTestCaseName.....	79	Client Application.....	86
PPTSVersion.....	77, 98	Protocol.....	1, 7, 9, 10, 13, 17, 27, 30, 40, 44, 56, 75, 143, 149, 155, 157
Pqw.....	75, 99	Protocol specified.....	9
PRD 2.0.....	1	use.....	9
Preliminary.....	86, 149, 152, 158, 161	Protocol traces.....	143, 157
Preliminary Verdicts.....	86, 149, 152, 158, 161	viewing.....	157
Preparing.....	99	Protocol Viewer.....	143, 155, 157
use PTSCControlClient.....	99	correlate.....	155
Presents...7, 9, 10, 17, 27, 30, 46, 53, 86, 99, 101, 108, 109, 124, 129, 143, 146		PTS.....	155
MMIs.....	86	Running.....	157
Press.....	13, 55, 65, 68, 109	starting.....	157
OK button.....	68	used to.....	157
Press Ctrl.....	109	using.....	157
Pressed.....	13, 45, 65, 70, 86, 135	Protocol Viewer application.....	143
Cancel button.....	86	Provides...53, 55, 56, 57, 58, 62, 65, 68, 71, 73, 86, 99, 107, 109, 131, 143, 155	
OK.....	65	PTS Control API.....	73
Primary windows.....	10	the User Interface.....	86
Proceeding in.....	135	Provides its own.....	58
Proceeding in order.....	135	PszDescription.....	86
Process by selecting.....	68	PszEntryName.....	84
Processed by.....	53, 68, 73, 75, 101, 107, 131	PszMessage.....	86
Product.....	1, 107, 108, 109, 121	PszNewParamValue.....	84
Product Details.....	109, 121	PszPathOfPtsFile.....	75, 101
Product Listings.....	1	PszPathOfWorkspace.....	75, 101

<i>PszProjectname</i>	77, 79, 84, 86	<i>PTS Endpoint Device</i>	44, 98, 101, 143, 155
<i>PszProjectName,wID</i>	86	<i>PTS Executable Test Suite</i>	9
<i>PszReponse</i>	86	<i>PTS executables</i>	9, 58, 67
<i>of</i>	86	<i>PTS Implicit Send DLL</i>	65
<i>PszResponse</i>	86	<i>PTS LE</i>	143
<i>of</i>	86	<i>PTS Program Settings</i>	46
<i>PszResponse,responseSize,L</i>	86	<i>PTS Protocol Viewer</i>	143, 152, 155, 157
<i>PszResponseBuffer</i>	86	<i>Starting</i>	157
<i>PszTestCase</i>	79, 86	<i>using</i>	143
<i>PszTestCaseDesc</i>	79	<i>PTS Protocol Viewer application</i>	157
<i>PszTestCaseName</i>	79, 86	<i>coupled</i>	157
<i>PszWorkspaceName</i>	75	<i>PTS removes</i>	62
<i>from</i>	75	<i>tags</i>	62
<i>PszWorkspacePath</i>	75	<i>PTS Report</i>	107, 108, 109
<i>PTCs</i>	56	<i>Creating</i>	109
<i>PTS</i> .. 1, 5, 7, 8, 9, 10, 17, 30, 35, 41, 44, 45, 46, 53, 55, 56, 57, 58, 62, 65, 67, 68, 70, 71, 72, 73, 74, 75, 77, 79, 84, 86, 98, 99, 101, 107, 108, 109, 121, 131, 143, 145, 146, 149, 152, 155, 157		<i>PTS supports</i>	17
<i>Automating</i>	53, 73, 131	<i>PTS Team</i>	67
<i>center</i>	146	<i>PTS Technical</i>	67, 70, 101, 152
<i>contains</i>	30	<i>PTS Technical Support</i>	67, 70, 101, 152
<i>corner</i>	143, 145	<i>PTS Terminology</i>	9
<i>During</i>	98	<i>PTS test case operation</i>	56
<i>Endpoint Device</i>	155	<i>PTS Test Scripting</i>	99
<i>exiting</i>	157	<i>PTS Test Suites</i>	86
<i>Implementation Under Test</i>	86	<i>PTS to the IUT</i>	149
<i>Implicit Send</i>	58	<i>PTS toolbar</i>	45
<i>imported</i>	1	<i>button</i>	45
<i>item</i>	1	<i>PTS User Interface</i>	77, 79, 84, 86
<i>IUT</i>	65	<i>corner</i>	86
<i>lines</i>	62	<i>window</i>	77, 79, 84
<i>name</i>	109	<i>PTS when the</i>	1
<i>number</i>	98	<i>PTS Workspace</i>	5, 108, 109, 131
<i>order</i>	57	<i>existing</i>	131
<i>part</i>	73, 99	<i>PTS.exe</i>	73
<i>Phone Book Access Profile</i>	149	<i>PTS_LOGTYPE logType</i>	86
<i>Protocol Viewer</i>	155	<i>PTS_LOGTYPE_END_TEST</i>	86
<i>release</i>	109	<i>PTS_LOGTYPE_ERROR</i>	86
<i>Select</i>	109	<i>PTS_LOGTYPE_EVENT_SUMMARY</i>	86
<i>starting</i>	101, 157	<i>PTS_LOGTYPE_FINAL_VERDICT</i>	86
<i>Use</i>	107, 108	<i>PTS_LOGTYPE_IMPLICIT_SEND</i>	86
<i>using</i>	57, 107, 109	<i>PTS_LOGTYPE_INFRASTRUCTURE</i>	86
<i>PTS Application</i>	8, 46, 152, 155, 157	<i>PTS_LOGTYPE_MESSAGE</i>	86
<i>Exit</i>	8	<i>PTS_LOGTYPE_PRELIMINARY_VERDICT</i>	86
<i>page of</i>	157	<i>PTS_LOGTYPE_RECEIVE_EVENT</i>	86
<i>PTS BR</i>	143	<i>PTS_LOGTYPE_SEND_EVENT</i>	86
<i>PTS clicking</i>	1	<i>PTS_LOGTYPE_START_TEST</i>	86
<i>PTS Control API</i> .. 73, 74, 75, 77, 79, 84, 86, 98, 99, 101		<i>PTSControl</i>	73, 75, 86
<i>connected</i>	79	<i>PTSControl.dll</i>	73
<i>indicates</i>	86	<i>PTSControl.h</i>	73
<i>object</i>	86	<i>PTSCONTROL_E_BLUETOOTH_ADDRESS_NOT_FOUND</i>	98, 101
<i>provides</i>	73	<i>PTSCONTROL_E_CLIENT_LOG_NOT_EXPECTED_TO_FAIL</i>	101
<i>use</i>	73		
<i>using</i>	73, 75, 99		
<i>PTS Control Client</i>	99		
<i>Test Scripts</i>	99		
<i>PTS Development Team</i>	41, 46		
<i>case</i>	41		
<i>PTS during</i>	107		
<i>PTS Endpoint</i>	44, 98, 101, 143, 155		

PTSCONTROL_E_FAILED_TO_CREATE_WORKSPACE
CE 101

PTSCONTROL_E_FAILED_TO_OPEN_WORKSPACE
..... 101

PTSCONTROL_E_FUNCTION_NOT_IMPLEMENTED
..... 79, 101

PTSCONTROL_E_GUI_UPDATE_FAILED 101

PTSCONTROL_E_ICS_ENTRY_NOT_CHANGED. 101

PTSCONTROL_E_ICS_ENTRY_NOT_FOUND..... 101

PTSCONTROL_E_ICS_ENTRY_UPDATE_FAILED 101

PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_ALREADY_REGISTERED 101

PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_NOT_EXPECTED_TO_FAIL 101

PTSCONTROL_E_IMPLICIT_SEND_CALLBACK_NOT_REGISTERED 101

PTSCONTROL_E_INTERNAL_ERROR 101

PTSCONTROL_E_INVALID_IXIT_PARAM_VALUE 101

PTSCONTROL_E_INVALID_PIXIT_PARAM_VALUE
..... 101

PTSCONTROL_E_INVALID_TEST_SUITE 101

PTSCONTROL_E_IXIT_PARAM_NOT_CHANGED 101

PTSCONTROL_E_IXIT_PARAM_NOT_FOUND..... 101

PTSCONTROL_E_IXIT_PARAM_UPDATE_FAILED
..... 101

PTSCONTROL_E_PICS_ENTRY_NOT_CHANGED
..... 101

PTSCONTROL_E_PICS_ENTRY_NOT_FOUND... 101

PTSCONTROL_E_PICS_ENTRY_UPDATE_FAILED
..... 101

PTSCONTROL_E_PIXIT_PARAM_NOT_CHANGED
..... 101

PTSCONTROL_E_PIXIT_PARAM_NOT_FOUND .. 101

PTSCONTROL_E_PIXIT_PARAM_UPDATE_FAILED
..... 101

PTSCONTROL_E_PROJECT_NOT_FOUND..... 101

PTSCONTROL_E_PROJECT_VERSION_NOT_FOUND
D..... 101

PTSCONTROL_E_PTS_FILE_FAILED_TO_INITIALIZE
..... 101

PTSCONTROL_E_PTS_VERSION_NOT_FOUND. 101

PTSCONTROL_E_TEST_SUITE_PARAM_UPDATE_FAILED 101

PTSCONTROL_E_TESTCASE_NOT_ACTIVE 101

PTSCONTROL_E_TESTCASE_NOT_FOUND 101

PTSCONTROL_E_TESTCASE_NOT_STARTED ... 101

PTSControlClient 73, 86, 98, 99, 101

 running..... 101

 using 99

PTSControlClient application..... 101

PTSControlClient Test Script..... 99

PTSLmplicitSendCallback 86

PTSPV..... 143, 155

PullBdAddr 98

Purchased..... 13

Purpose..... 9, 40, 46, 62, 108, 126, 149, 152, 155

 Displaying..... 40

Q

QDID 109

QLI..... 75, 107

Qualification Listing Interface 107, 109

Qualification test evidence 107

Qualified Device Listing..... 1

Qualified Listing Interface..... 75

QueryInterface..... 86

R

Receives 56, 75, 77, 79, 86, 98, 147, 149, 152

 BDADDR 98

Recent Workspaces 5, 109

Refer 13, 17, 46, 58, 86, 99, 107, 108, 132

 IPTSLmplicitSendCallbackEx 86

RegisterImplicitSendCallback 86, 101

RegisterImplicitSendCallbackEx 86

Regsvr32 PTSControl..... 101

Release..... 58, 67, 77, 86, 98, 109

 PTS..... 109

Remove Checked Items button..... 127, 130

 clicking..... 127

Removing..... 9, 13, 70, 137

 checkmark 70

 project..... 9

 Test Case..... 137

Replace 55, 62, 70, 71, 72, 86

 used..... 55

Replaced with the name of..... 62

Report 1, 41, 107, 108, 109, 121, 124, 129, 149

 Contents 108

 of creating..... 109

 time to create 109

Report Generator menu 121

Represent 1, 8, 9, 17, 30, 77, 98

 used..... 8

ResponseSize..... 86

Retrieves..... 98

 Bluetooth Device Address 98

Retry..... 65

Return Values 58, 65, 72, 75, 77, 79, 84, 86, 98

Returning 65, 70, 75, 86, 101

 HRESULT..... 75

 NULL 65, 86

 OCTETSTRING 101

 Project Settings..... 70

Right..... 1, 8, 10, 17, 41, 50, 68, 79, 86, 101, 109, 121, 129, 133, 137, 139, 143, 145, 149, 153

Right clicking..... 50, 143, 145

Room for 86

 NUL 86

Run	27, 41, 44, 46, 50, 53, 55, 73, 74, 75, 99, 101, 109, 129, 131, 135, 145, 146, 152, 153, 155, 157
Protocol Viewer	157
PTSControlClient	101
Test Case	135
Test Script	101
Run Script	135
Running during	109
RunTestCase	79, 101

S

S is s/n	79
S/n	77, 79
Sample	58, 71, 73, 86, 98, 99
Sample Program	86, 98, 99
Sample Source Code	58, 71
Save	46, 109, 121, 143, 157
Scoping	65
Script	53, 73, 99, 131, 132, 133, 135, 137, 139, 141
Script Tool Window	132, 133, 135, 137
select	132
Script Tool Window icon	132
toolbar	132
Scripted Operation	53, 73, 131
Scripting	53, 73, 99, 132, 135
Search	1, 17, 143, 149
Search string	17
Secure Simple Pairing	65
Segment Preliminary	158, 161
Select1	5, 7, 8, 9, 10, 13, 17, 30, 35, 40, 41, 46, 53, 58, 65, 68, 73, 77, 79, 84, 86, 99, 101, 108, 109, 121, 124, 126, 127, 129, 130, 131, 132, 133, 135, 137, 139, 141, 143, 145, 147, 149, 152, 153
used	1, 58, 129
Select	133
Select	133
Select	157
Select All	13, 109, 127
Select Duplicates	129
use	129
Select Duplicates button	129
Select Latest	109, 129
Select Test Cases	79, 101, 109, 127, 129, 135
Selected device description	121
Selected during	108
Selecting1	5, 13, 58, 109, 127, 129, 131, 132, 141, 152
events	152
Implementation Under Test	1
individual test case results	127
Legacy	5
older test results	129
PTS	109
Script Tool Window	132
Test Case	141
Test Cases	131

Selection	1, 10, 13, 17, 30, 50, 65, 109, 126, 127, 135, 152, 153
Description	10
page allows	152
Send	55, 56, 57, 62, 70, 141
Send event	86, 147, 149, 152
Sending	46, 86, 141, 147, 149
Test Script	141
Server	53, 74, 149
Service Class	1
Set	13, 30, 41, 46, 53, 57, 62, 71, 73, 75, 84, 86, 101, 108, 109, 131, 135, 152, 158, 161
pbResponselsPresent	86
PICS	75
Test Cases	135
TRUE	84, 86
SetControlClientLoggerCallback	86
Setting adds communication	46
Setting pbResponselsPresent	86
FALSE	86
TRUE	86
Settings	7, 30, 35, 41, 46, 50, 53, 68, 108, 109, 131, 147, 152, 153
Settings affects	46
Shift	13
Shortcut	13, 109, 157
Should always	30
Show	27, 44, 79, 109, 145, 149
OBEX	149
Show Purpose	79
ShowTag	62
SIG	53
Single test case	41
Executing	41
Software build requirements	67
Stack	56, 155
Standard	9, 41, 46, 57, 58, 77, 98, 152
Start	1, 17, 35, 41, 44, 58, 62, 67, 68, 74, 86, 98, 99, 101, 109, 135, 139, 146, 147, 149, 157
Protocol Viewer	157
PTS	101, 157
PTS Protocol Viewer	157
Test Case	86
Start Page	35
Start PTS	74, 101
Start Test Case	86, 147, 149
Std	57, 58, 67
STL	57
Stop	1, 109, 135
Stop executing	135
Stop Searching	1
Stopping	135
Test Script	135
StopTestCase	79
Straightforward	109

-
- String58
 - strPrompt58
 - StrMmiText 58, 62, 65
 - beginning62
 - contents65
 - of 62, 65
 - StrPrompt58
 - string58
 - StrPrompt,MMI_Style_Edit158
 - StrTestCaseName58
 - Style58, 62, 65, 75, 86
 - MMI86
 - used to select58
 - Style Name86
 - Subfolder 1, 75
 - name75
 - Suite10, 13, 17, 35, 62, 72
 - SzLogType86
 - SzTime86
 - T**
 - Tag62, 72, 86, 99, 155
 - Implicit Send DLL display62
 - PTS removes62
 - Take quite 109
 - long time 109
 - TC_CLIENT_ABC_BV_01_I9
 - TC_COD_BV_01_299
 - TC_COD_BV_01_I99
 - TC_SERVER_OPH_BV_03_I62
 - TC_SERVER_OPH_BV_07_I62
 - TCP71
 - Team67
 - Template Library57
 - Test case...9, 10, 27, 30, 35, 40, 41, 44, 46, 50, 53, 56, 58, 62, 65, 68, 70, 72, 73, 75, 79, 86, 99, 101, 108, 109, 124, 126, 127, 129, 130, 131, 133, 135, 137, 139, 141, 143, 145, 146, 147, 149, 152, 153, 155, 158, 161
 - Adding 133, 141
 - Choose 133
 - clicking79
 - containing79
 - during86
 - executing 62, 86
 - Highlight 139
 - If 135
 - names 131
 - number99
 - of... 27, 50, 56, 79, 86, 108, 109, 143, 145, 146, 149, 152, 153
 - Removing 137
 - running 135
 - select 131, 141
 - set 135
 - start86
 - When44
 - Test Case Ended 149, 155
 - Test case execution... 41, 44, 56, 58, 86, 108, 109, 124, 126, 143, 145, 146, 149, 153, 155, 158, 161
 - Aborting44
 - of 86, 109, 143, 149
 - Test case execution log41, 86, 126, 143, 145, 146, 149, 153, 155
 - Viewing126
 - Test Case has encountered86
 - Test Case History 109, 124, 145
 - Test Case History Tool Window124, 145
 - Test Case History Tool Window displays 124
 - Test Case History window145
 - Test Case Naming9
 - Test case results27, 108, 109, 127, 129, 130
 - Deleting130
 - Test case runs for109
 - Test Case Started147, 155
 - Test Case View 27, 50, 153
 - of50, 153
 - Test cases produce41
 - number41
 - Test Component56
 - Test Control Notation1143
 - Test Execution Log... 108, 109, 126, 143, 146, 149, 152
 - Test History124
 - Editing124
 - Test history editor124
 - Test Plan Generator 1, 30, 75, 107, 109
 - Using107
 - Test Plan Generator's list109
 - Test Procedure53
 - Test Purpose9, 79
 - corresponding79
 - Test Purposes vs9
 - Test run 124, 158, 161
 - Test Script..99, 101, 131, 132, 133, 135, 137, 139, 141, 143, 149
 - added137
 - Create99
 - Editing137
 - Executing135
 - name101
 - PTS Control Client99
 - Running101
 - Select133
 - Send141
 - Stopping135
 - Test Script Window135, 139
 - Test Suite...9, 10, 13, 17, 35, 46, 50, 62, 68, 70, 71, 72, 77, 108, 131, 133, 143, 145, 147, 152
 - inclusion13
 - Test Suite Selector7, 10, 17
 - Using10
 - Test Suite Selector dialog10, 17
 - Test Suite Selector will display10
 - of10
 - Testcase 79, 99, 109
 - TestCaseView 77, 79, 84
-

Tester Information.....	109
Testing.....	9, 139
Profile Implementation Extra Information	9
TestReport.....	109
TestScriptSample.....	99
TestScriptTemplate.....	99
Testsuite.....	99
Text... 10, 17, 58, 62, 68, 75, 86, 99, 109, 121, 143, 147, 149	
Text file.....	99
used to create.....	99
Text string.....	75
That has been encrypted.....	108
That is... 1, 30, 44, 46, 50, 53, 55, 58, 65, 68, 70, 71, 73, 75, 77, 79, 84, 86, 98, 99, 107, 121, 129, 137, 143, 147, 155	
That must.....	9, 30, 86
The first... 1, 5, 13, 30, 35, 41, 44, 46, 65, 75, 79, 86, 99, 108, 109, 135, 139, 141, 143, 147, 149, 155	
The instructions.....	65, 99, 141
the user.....	65
The IUT..... 9, 27, 30, 41, 44, 45, 53, 62, 65, 70, 86, 149	
The language	58
The list... 1, 10, 13, 17, 30, 65, 109, 121, 124, 126, 127, 147	
The TTCN.....	143, 149
The TTCN operating.....	143
The user	65, 70, 74, 86, 109
the instructions	65
The User Interface.....	74, 86
provide.....	86
This feature	53, 73, 131
This functionality	46, 62, 67, 86
Time.... 1, 5, 7, 9, 13, 30, 45, 53, 55, 58, 70, 73, 75, 86, 107, 108, 109, 131, 133, 135, 141, 145, 146, 147, 149, 153	
create.....	109
Time consuming.....	55
Time logging.....	50
Time to.....	1, 107, 109, 135
Time to create	109
report.....	109
Times during	149
To execute.....	1, 27, 56, 135, 158, 161
To indicate.....	65, 149
Toolbar	30, 35, 44, 132
Script Tool Window icon	132
TP.....	9
TPG.....	1, 7, 30, 75, 107
Using.....	107
Tree where	27
True.....	30, 58, 121, 146
set	84, 86
setting pbResponselsPresent.....	86

True during.....	30
TRUE if.....	86
TRUE indicating.....	71
TSPC_ALL	30
TSPC_HFP15_2_3.....	30
TSPX_bd_addr_iut.....	35
TSPX_use_implicit_send	71
value.....	71
TTCN.....	143, 149
Txt	62
Txt file.....	62
Type	35, 50, 58, 65, 70, 86, 109, 143, 152, 155

U

U in Project	79
UINT	58, 77, 79
UINT iProject.....	77
UINT iTestCase	79
UINT mmiStyle.....	58
UllBthAddr.....	75
ULONGLONG.....	75, 98
ULONGLONG ullBthAddr.....	75
Un.....	13
Unattended operation.....	73, 86, 131
Uncheck the.....	30
Uncheck the checkbox	30
Unchecking	109
checkbox	109
Unicode	75, 77, 79, 84, 86
Unicode string.....	77, 79, 86
Unicode UTF.....	75
Unloading.....	58
DLL	58
UnregisterImplicitSendCallback	86, 101
UnregisterImplicitSendCallbackEx.....	86
UnSelect All	109
Unselected.....	129
Until the test.....	153
Until the test case.....	153
UpdateICS	84, 101
UpdateIXITParam.....	75, 84, 101
UpdatePics.....	84, 101
calling	101
UpdatePixitParam.....	75, 84, 101
using.....	75
Updating.....	46, 101
IXIT	101
PICS	101
PIXIT.....	101
Upload Report.....	109
Upload Report requires	109
Usage Notes	70

-
- Use 9, 50, 53, 57, 58, 62, 67, 68, 70, 72, 73, 75, 86, 99, 101, 107, 108, 109, 121, 124, 129, 131, 143
 Debug75
 DllMain58
 Implicit Send86
 IPTSImplicitSendCallbackEx86
 Profile Tuning Suite109
 protocol specified9
 PTS107, 108
 PTS Control API73
 Select Duplicates129
 Visual Studio Debugger67
 wcscpy_s86
 Use OpenWorkspace75
 Use PTSControlClient99
 Preparing99
 Used .. 1, 5, 8, 10, 13, 17, 27, 30, 35, 41, 46, 50, 53, 55, 56, 57, 58, 62, 65, 67, 68, 70, 71, 73, 75, 77, 79, 86, 98, 99, 101, 107, 108, 109, 121, 124, 129, 131, 147, 149, 152, 157, 158, 161
 consolidate107
 create86, 99, 109
 develop73
 direct107
 distinguish109
 enable71
 implement56
 indicate27, 86
 instruct65
 replace55
 represent8
 select1, 58, 129
 Workspace99
 Used as13, 71, 73, 77, 79, 149
 Used during107
 Used for this purpose45, 109
 Used in .. 1, 17, 30, 41, 53, 55, 58, 62, 73, 86, 109, 129, 131, 149
 Used in tracking41
 Used to ...1, 8, 17, 27, 30, 35, 46, 50, 53, 55, 56, 58, 65, 70, 71, 73, 75, 77, 86, 99, 107, 109, 124, 127, 129, 149, 152, 157, 158, 161
 Protocol Viewer157
 Used to consolidate107
 Used to create86, 99, 109
 text file99
 Used to develop73
 Used to direct107
 Used to distinguish109
 Used to enable46, 71
 Used to hide one17
 Used to implement56
 Used to indicate27, 86, 149
 Used to instruct65
 Used to replace55
 Used to represent8
 project8
 Used to select1, 58, 129
 style58
 User Action Requested70
 User Action Required70
 User Defined Implicit Send DLL50, 88
 User Interface74, 86
 Using ..1, 7, 8, 10, 30, 35, 44, 46, 56, 57, 58, 68, 70, 72, 73, 75, 77, 79, 86, 99, 107, 109, 121, 129, 131, 139, 141, 143, 152, 157
 dropdown list121
 IUT70
 Main Test Component56
 PICS30
 PIXIT35
 Protocol Viewer157
 PTS57, 107, 109
 PTS Control API73, 75, 99
 PTS Protocol Viewer143
 PTSControlClient99
 Test Plan Generator107
 Test Suite Selector10
 TPG107
 UpdatePixitParam75
 Using Implicit Send73, 86, 131
 Using PTS30, 157
 Using Windows Explorer8
V
 Value1, 35, 41, 44, 58, 65, 71, 75, 77, 79, 84, 86, 98, 101, 147, 149
 0x000075, 98
 GetProjectName101
 GetTestCaseName101
 logType86
 PICS item84
 TSPX_use_implicit_send71
 Value represents101
 general101
 Verdict Description86, 143, 147, 149, 155
 Verdict Determination149, 158, 161
 Version.1, 10, 30, 41, 57, 58, 62, 67, 71, 77, 86, 98, 99, 101, 108, 147
 Viewing126, 157
 protocol traces157
 test case execution log126
 Virtual Sniffer155
 Visual67, 71, 73, 86
 Visual Studio67, 71, 73
 needed67
 Visual Studio 200867
 Visual Studio Debugger67
 use67
W
 Wcscpy_s86
 Use86
 When1, 10, 30, 41, 44, 56, 57, 58, 65, 67, 68, 70, 72, 73, 74, 75, 77, 79, 86, 99, 101, 107, 108, 109, 121, 124, 126, 129, 135, 143, 146, 149, 152, 153, 158, 161
 test case44
 When a Test Case44, 86
 When developing73

When mixing	67	Windows LoadLibrary.....	72
When pbResponselsPresent.....	86	With 9, 17, 27, 30, 35, 40, 41, 53, 55, 56, 57, 58, 62, 65,	
When reviewing.....	146	67, 68, 70, 73, 74, 75, 77, 79, 84, 86, 98, 99, 101,	
When selecting.....	121, 126	107, 108, 109, 121, 131, 139, 143, 145, 147, 152,	
item	126	155, 157	
When testing	30	name of	75
Which may.....	1, 75	With existing.....	86
WID.....	86	With other.....	70, 152
WID values.....	86	With other events.....	152
WideCharToMultiByte	75	With text	109
Will initiate	1	With the name of.....	62, 75
Will result.....	158, 161	WORD wID	86
Win32 BOOL	86	Workspace ... 1, 5, 7, 8, 9, 10, 13, 27, 30, 35, 40, 44, 50,	
WINAPI.....	58	53, 62, 68, 70, 73, 75, 77, 79, 84, 99, 101, 109, 131,	
WINAPI ImplicitSendPinCode	58	133	
WINAPI ImplicitSendStyle	58	create.....	75, 101
WINAPI ImplicitStartTestCase	58	Creating.....	99
WINAPI ImplicitTestCaseFinished	58	Deleting	8
Window		existing	75, 99
PTS User Interface.....	77, 79, 84	name.....	75
Window contains.....	10, 145	Opening/Creating.....	75
Windows.....	7, 10, 13, 17, 30, 35, 50, 57, 58, 70, 72, 73,	used.....	99
74, 75, 77, 79, 84, 86, 98, 109, 132, 143, 145, 146,		Workspace file	75
157		name.....	75
copied	109, 143	Workspace Tool Window.....	27, 40, 44, 68
Windows API.....	58, 74, 75	Workspace window.....	133
calling.....	74	Wprintf.....	77, 79
Windows DLL	50		
Windows Dynamic Link Libraries	57, 73	X	
Windows Explorer	145, 157	Xlsx.....	99
Windows Explorer window.....	145	XML	99, 101, 107, 108, 109, 143
cause	145	XML Notepad	99
		Y	
		Your Suite	7, 10, 13