17 Scripts for Filtering, Analyzing, and Visualizing

Requires: AppTransaction Xpert Advanced Capabilities module

Use the scripting feature to create scripts that can manipulate a Transaction Analyzer model file (for example, by deleting certain packets), perform analyses and display custom statistic graphs, or create custom visualizations. Scripts use a Python API for interacting with Transaction Analyzer model files.

Use scripts to

- Create a balloon label on all packets or messages meeting certain criteria of interest
- Report on the number of connections in a trace
- Report on the HTTP “user agent”
- Exclude all FINs
- Perform sanity checks, such as checking for a large number of missing packets

The Advanced menu in the Transaction Analyzer window provides access to scripting. From the Advanced menu, you can open the scripting console, run built-in scripts, and create/edit/run custom scripts.

For more information about scripting, look in the OPNETWORK 2010 Proceedings for session 1443 (Customizing Analysis and Visualization in ACE™ Analyst). You might also find session 1396 (Introduction to the Python Programming Language) useful.

Related Topics

- *AppTransaction Xpert Scripts*
- *Scripting Console*
- *Scripting API*

Scripting Console

Requires: AppTransaction Xpert Advanced Capabilities license

Access: Transaction Analyzer menu bar: Advanced > Scripting Console…

The scripting console provides a way to quickly execute Python commands. You can use the console to run short Python scripts that don't need to be saved, to test commands before adding them to a script in the script editor, or to view help for a Python command or module.

Procedure 17-1 Executing Python Commands in the Scripting Console

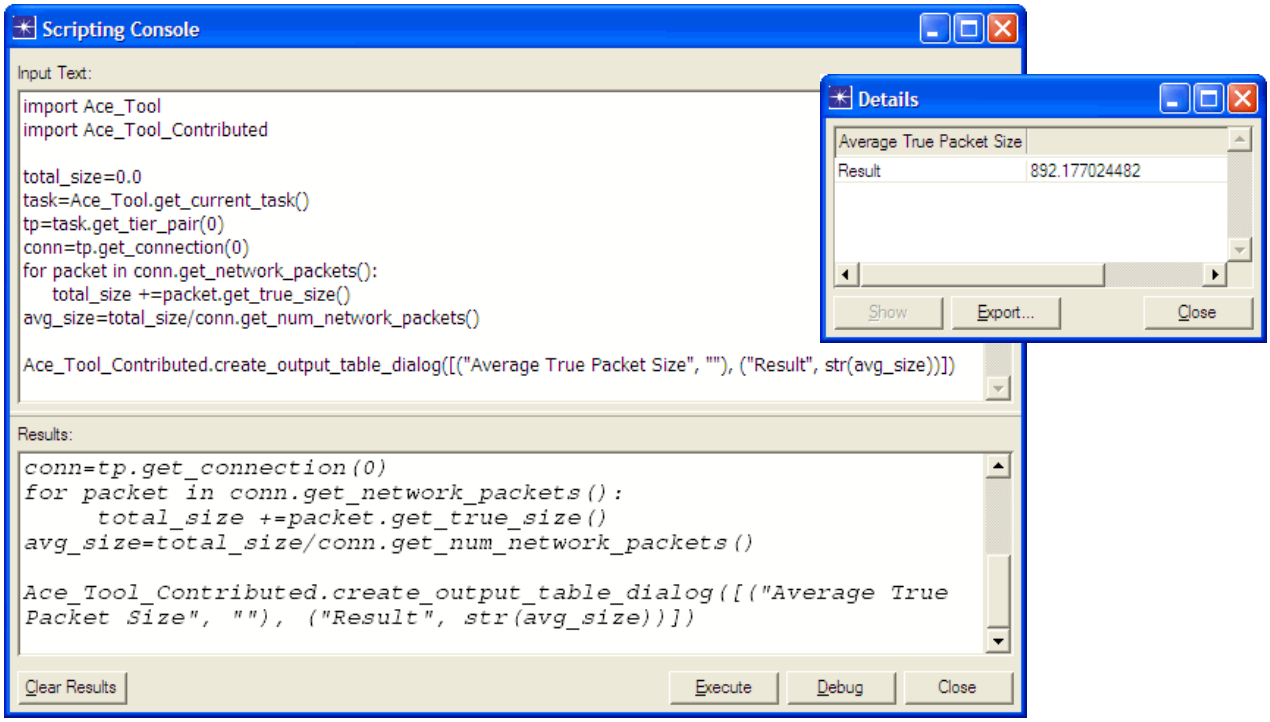
1. Open the scripting console (Advanced > Scripting Console…).
2. Type the commands into the Input Text pane.
   - The commands should begin by importing any Python modules that are needed. Generally, this is:

     ```
     import Ace_Tool
     import Ace_Tool_Contributed
     ```

   - To produce output, include Python *print()* statements or use output commands such as *Ace_Tool_Contributed.create_output_table_dialog()*.
3. Click the Execute button to run the commands.

   ➥ The commands are echoed in the Results pane, along with any print output or error message.

Figure 17-1 Command Execution in Scripting Console



End of Procedure 17-1

Procedure 17-2 Debugging Python Commands in the Scripting Console

1. Debugging uses an application console window. Choose Edit > Preferences and check the setting of the “console” preference (in the Miscellaneous section). If this preference is set to FALSE, change it to TRUE and then restart AppTransaction Xpert.
2. Open the scripting console (Advanced > Scripting Console…).
3. Type the commands into the Input Text pane, or copy them from a script editor.
   - Remember to import any Python modules that are needed. Typically, you add the following lines at the beginning:

     ```
     import Ace_Tool
     from Ace_Tool_Contributed import ace_debug
     ```

   - Add Python *print()* statements to display intermediate values or other useful debugging information.
   - Add the *Ace_Tool_Contributed.ace_debug()* statement at some useful place. (This statement runs the Python debugger in the console window. The debugger supports inspecting variables, single-step execution, and breakpoints.)
4. Click the Debug button.

   ➥ The commands are echoed in the Results pane. Print output, error messages, and debugger prompts appear in the console window.

End of Procedure 17-2

Procedure 17-3 Viewing Python Module Help

1. Open the scripting console (Advanced > Scripting Console…).
2. Import the module of interest:

   ```
   import <module_name>
   ```

3. Type the command:

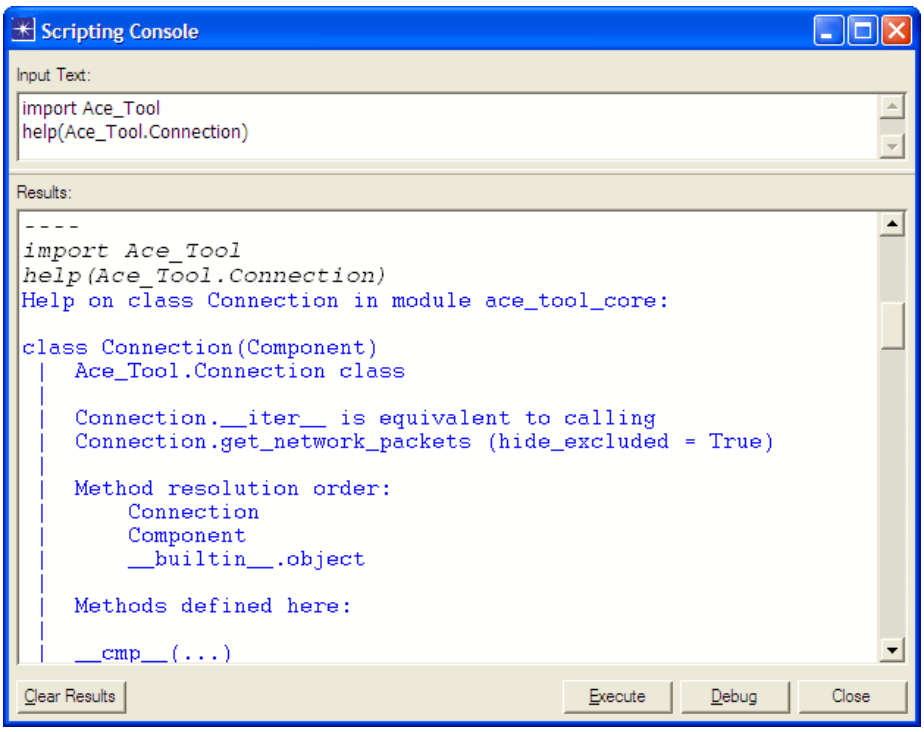
   ```
   help (<topic>)
   ```

   where <topic> is the name of a Python module (such as “Ace_Tool”), class, method, or function.

4 Click the Execute button.

➥ The help is displayed in the Results pane.

Figure 17-2 Help in Scripting Console



End of Procedure 17-3

Related Topics

- *AppTransaction Xpert Scripts*
- *Scripting API*

AppTransaction Xpert Scripts

AppTransaction Xpert scripts are based on the Python programming language. Python is object-oriented and integrates well with C and C++.

For more information about Python, visit the Python web site (www.python.org).

Script Editor

You create and edit scripts in an edit pad (called an script editor when used for Python scripts).

Procedure 17-4 Creating a Script

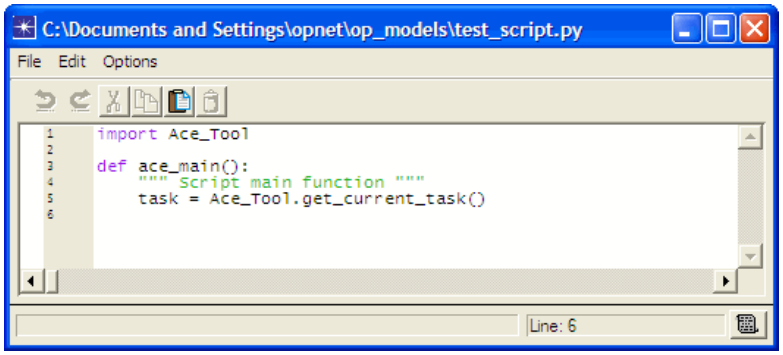
1 In the Transaction Analyzer window, choose Advanced > New Script…

➥ A file browser opens.

2 In the file browser, enter a name for the script, navigate to the directory in which to save the script, and click OK.

➥ A script editor opens with the initial statements for a script.

Figure 17-3 Script Editor



3 Enter the script in the script editor. When finished, choose File > Save.

End of Procedure 17-4

Procedure 17-5 Editing a Script

1 In the Transaction Analyzer window, choose Advanced > Edit Script…

➥ A file browser opens.

2 In the file browser, select the script to edit and click OK.

➥ A script editor opens with the selected script.

3 Edit the script in the script editor. When finished, choose File > Save.

End of Procedure 17-5

Procedure 17-6 Running a Script

1 In the Transaction Analyzer window, choose Advanced > Run Script…

➥ A file browser opens.

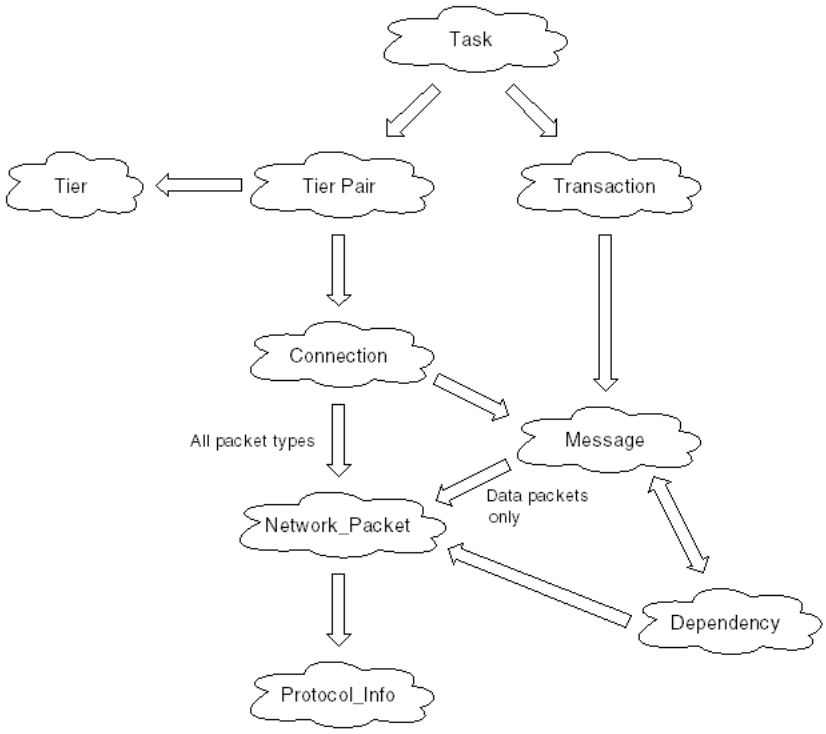
2 In the file browser, select the script to run and click OK.

End of Procedure 17-6

Traversing the Object Hierarchy

The following figure shows the hierarchy of objects in a Transaction Analyzer model that are accessible using the scripting API.

Figure 17-4 Object Hierarchy



A common way of stepping through all packets in a Transaction Analyzer model is to use nested `for` loops. This approach gives you control over the order in which you search through the packets and lets you collect statistics in a structured manner. For example:

```
# Get the current task
task = Ace_Tool.get_current_task()
# Iterate through all packets of all connections of all tier_pairs
for tp in task.get_tier_pairs():
   for conn in tp.get_connections():
      for pkt in conn.get_network_packets():
         # Do something with the packets...
```

An alternate way of stepping through packets is by message. Note, however, that application messages do not contain all packets (for example, dataless ACKs are not included in messages). For example:

```
# Get the current task
task = Ace_Tool.get_current_task()
# Iterate through all messages of all transactions
for transaction in task.get_transactions():
   for msg in transaction.get_messages():
      for packet in msg.get_network_packets():
         # Do something with the packets...
```

Iteration Hierarchy

The *__iter__* method is defined in many classes to allow a simple way of traversing a frequently used hierarchy. This hierarchy is

- Task
- Tier_Pair
- Connection
- Network_Packet
- Protocol

For example, to iterate the included connections in a tier pair, you can write

```
for connection in a_tier_pair:
```

instead of

```
for connection in a_tier_pair.get_connections(True):
```

Note—The *__iter__* methods are defined to operate only on included objects. If you need to iterate over all objects, including those currently excluded, you must use the longer version. For example, to iterate all connections in a tier pair, use

```
for connection in a_tier_pair.get_connections():
```

You can mix these two approaches. For example, to step through all packets of included messages:

```
# Get the current task
task = Ace_Tool.get_current_task()
# Iterate through included messages of included transactions
for transaction in task.get_transactions(True):    # get included transactions
   for msg in transaction:                         # get included messages
      for packet in msg.get_network_packets():     # get all packets
         # Do something with the packets...
```

Filtering with Scripts

Filtering an application task (Transaction Analyzer model) file can improve analysis results by

- removing extraneous traffic
- letting you focus on specific transactions, protocols, or hosts

AppTransaction Xpert contains a variety of built-in filtering approaches, as described in Chapter 9 Filtering Traffic on page ATX-9-1. In addition, you can use scripting to supplement the built-in filtering by performing custom filtering on a task. Common elements of a script used for filtering include

- *include()*, *include_all()*—methods used to control which components of a model (packets, protocols, and so on) will be considered during analysis
- *exclude()*, *exclude_others()*, *delete()*, *delete_others()*—methods used to control which components of a model will not be considered during analysis
- *is_included()*, *is_excluded()*—methods used to test whether a component is included or excluded; available in the Connection, Message, Network_Packet, Tier_Pair, and Transaction classes
- *hide_excluded*—argument used by various methods to specify whether the iterator object that is returned should point to all components or only those that are not excluded (see *connection.get_messages()* for an example)

Figure 17-5 Filtering Example

```
import Ace_Tool
import Ace_Tool_Contributed

def ace_main():
   task = Ace_Tool.get_current_task()

   # Prompt user for the protocol to include
   prompt = "Protocol to be included (all others will be excluded):"
   protocol_name = Ace_Tool.get_user_string_input (prompt, "Include protocol")

   # Did the user actually give a protocol name (not press cancel)
   if protocol_name:
      packet_list = Ace_Tool_Contributed.get_protocol_packets (protocol_name)

      task.include (packet_list)
      task.exclude_others (packet_list)
```

Graphing with Scripts

AppTransaction Xpert can graph statistics that measure network and application performance within each tier pair and connection, as described in Chapter 16 Viewing Statistics on page ATX-16-1. You can use scripting to supplement the built-in graphing options.

Figure 17-6 Graphing Example

```
import Ace_Tool
import Ace_Tool_Contributed

def ace_main():
   """Creates network throughput graphs of specific protocols"""

   # Create an empty list for storing our graph data.
   # (The '[]' creates an empty list).
     graph_list = []

   # Create a graph showing HTTP throughput. We'll do this in three steps:
   #  1) Get a list of network packets, where all those packets in are in connections
   #     whose highest protocol is HTTP.
   #  2) Create the graph data "tuple", which includes the graph labels as well as
   #     the packets from step 1.
   #     -- This is a three-item Python "tuple" (where a tuple is created by putting
   #        parentheses around the comma separated items).
   #  3) Add the graph "tuple" to our graph list.

     http_packets_list = Ace_Tool_Contributed.get_protocol_packets ("HTTP")

  # Step 1: get the packet list for HTTP
     http_graph_tuple = ("Time (sec)", "HTTP Network Throughput(Kbps)",
                         http_packets_list)

  # Step 2: create the graph data "tuple"
     graph_list.append (http_graph_tuple)

  # Step 3: add the "tuple" to our graph list

    # Create a list of network packets in all connections and add it to our graph list.
    # Follow the same three steps as above, but for step 1 get ALL packets.
     all_packets_list = Ace_Tool_Contributed.get_network_packets ()
     overall_graph_tuple = ("Time (sec)", "Overall Network Throughput(Kbps)",
                            all_packets_list)
     graph_list.append (overall_graph_tuple)

    # Create and show network throughput graph,
    # using our list of graphs that we built up.
    Ace_Tool_Contributed.create_network_throughput_graph ("Network Throughput Graph",
                                                          graph_list, bucket_width=500)
```

Visualizations with scripts

AppTransaction Xpert visualizations apply different coloring and labeling schemes to the Data Exchange Chart or Tree View. You can apply different visualizations to highlight different types of information in a transaction. AppTransaction Xpert contains a variety of built-in visualizations; you can use scripting to write additional custom visualizations.

To apply the built-in visualizations, choose View > Visualization. Some choices in the list of built-in visualizations are labeled “(example script)”. You can find the Python scripts that implement these visualizations in the `<reldir>\sys\lib` directory and can use them as guides for writing your own visualizations.

Creating a Custom Visualization

A visualization is defined by two types of files:

- An ETS configuration file that links the visualization to the Transaction Analyzer menu
- One or more Python files that define the colors and labels for the visualization

After creating these files, refresh the model directories, and reopen the Transaction Analyzer model file, the defined visualization will appear in the Visualization menu.

Configuration File The configuration file defines the visualization name and the function names that implement the visualization. The file must have a name ending with `.ace.viz.ets` and must be placed in a model directory (the configuration files for built-in visualizations are in `<reldir>\sys\configs`).

Configuration files contain several sections, as follows:

```
#-----------------------------------------------------------------------
#
# Configuration file format
#
#-----------------------------------------------------------------------

start_visualization
   title:          <name to appear in menu>
   coloring:       <name of coloring section for this visualization>
   labels:         <name of label section for this visualization>
   library:
   init_function:
end_visualization

start_coloring
   title:              <name of coloring section>
   library:            <name of Python file containing coloring code>
   init_function:      <name of callback function for initializing coloring>
   appl_msg_function:  <name of callback for coloring messages>
   netpk_function:     <name of callback for coloring packets>
end_coloring

start_label
   title:              <name of label section>
   library:            <name of Python file containing label code>
   init_function:      <name of callback function for initializing labels>
   appl_msg_function:  <name of callback for labeling messages>
   netpk_function:     <name of callback for labeling packets>
end_label
```

You can define multiple visualizations in the same configuration file by repeating the different sections.

Python File The Python file implements the callback functions defined in the configuration file. The file must be placed in a model directory. Refer to the code for built-in visualizations (in the `<reldir>\sys\lib` directory) as a guide for creating custom visualizations.

Scripting API

The scripting application programming interface (API) is a set of classes, methods, properties, and functions that support the creation of scripts.

You can use the API to

- Iterate over objects such as tier pairs, connections, and packets
- Obtain information from the objects, such as:
  - Protocol attributes (for example, TCP Sequence numbers)
  - Information that AppTransaction Xpert calculates for every packets (for example, whether or not a packet is a retransmission)
  - The ASCII decode
  - Binary packet data
- Operate on objects (exclude, delete, and so on)
- Obtain information from the user
- Show calculated results within AppTransaction Xpert (for example, add a time-varying graph)

The classes and functions of the scripting API are contained in a module named Ace_Tool.

Additional classes and functions are contained in the Ace_Tool_Contributed Module.

Ace_Tool Classes

The Ace_Tool module defines several classes used to create logic scripts. The following figure shows the class hierarchy. Table 17-1 lists the classes and provides a link to a full description of each.

Figure 17-7 Ace_Tool Class Hierarchy

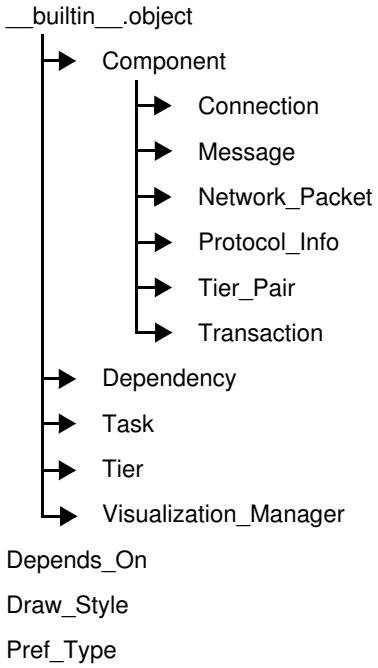


Table 17-1 Summary of Ace_Tool Classes

Class	Description
Component Class	Base class for application components.
Connection Class	Provides access to connections.
Dependency Class	Provides access to dependencies.
Depends_On Class	Enumerator class defining constants that specify dependency types.
Draw_Style Class	Enumerator class defining constants that specify draw style options for graphs.
Message Class	Provides access to and control of messages.
Network_Packet Class	Provides access to and control of packets (frames).
Pref_Type Class	Enumerator class defining constants that specify data types for input parameter dialog boxes.
Protocol_Info Class	Provides access to the protocol layers of a packet.

Table 17-1 Summary of Ace_Tool Classes (Continued)

Class	Description
Task Class	Provides access to and control of tasks.
Tier Class	Provides access to tiers.
Tier_Pair Class	Provides access to tier pairs.
Transaction Class	Provides access to transactions.
Visualization_Manager Class	Provides access to and control of visualizations.

Ace_Tool Functions

The Ace_Tool module defines some support functions you can use in scripts. These functions are

- *create_graph()*
- *get_current_task()*
- *get_time_bounds()*

create_graph()

Abstract Opens a graph dialog box with one or more traces.

Syntax Ace_Tool.create_graph(title, trace_list)

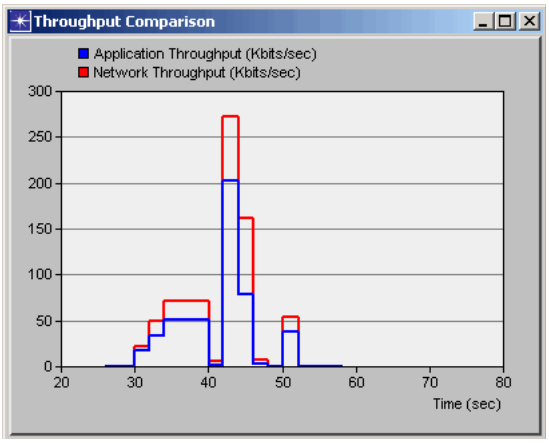
Argument	Type	Description
title	string	Title for the dialog box.
trace_list	list of tuples	One or more traces to be displayed in the graph (see Details).

Return Type No return value.

Example

```
app_thruput = ("Time (sec)", "Application Throughput (Kbits/sec)",
   Ace_Tool.Draw_Style.SQUARE_WAVE,[(26,0),(30,16.9),(32,33.5),(34,50.4),(40,1.7),
  (42,203.1),(44,78.8),(46,2.7),(48,0),(50,38.3),(52,0),(58,0)])
ntwk_thruput = ("Time (sec)", "Network Throughput (Kbits/sec)",
   Ace_Tool.Draw_Style.SQUARE_WAVE,[(26,0),(30,21.9),(32,49.4),(34,71.4),(40,5.7),
  (42,273),(44,161.1),(46,6.6),(48,0),(50,54.3),(52,0),(58,0)])
Ace_Tool.create_graph("Throughput Comparison", [app_thruput, ntwk_thruput])
```

Figure 17-8 Example Graph Dialog Box



Details This function opens a graph dialog box and draws one or more traces in it. Each trace is defined by a tuple in *trace_list*. The structure of *trace_list* is

```
[(x_title, y_title, draw_style, [(x_value, y_value)...])...]
```

The items comprising *trace_list* are

Table 17-2 trace_list Components

Item	Type	Description
x_title	string	Name for the x axis.
y_title	string	Name for the y axis.
draw_style	int	Type of trace to draw; must be one of the draw styles defined in the Draw_Style Class.
(x_value, y_value)	tuple	Tuple containing two floats, representing the x,y values for one point on the trace. The trace is defined by a list of such tuples, one for each point.

get_current_task()

Abstract Returns the current task for the active AppTransaction Xpert session.

Syntax Ace_Tool.get_current_task()

Argument	Type	Description
		no arguments

Return Type Task The current task.

Example

```
task = Ace_Tool.get_current_task()
tmp_list = []
for tp in task.get_tier_pairs():
   for conn in tp.get_connections():
      for npk in conn.get_network_packets ():
         tmp_list.append (npk)
```

Details None.

get_time_bounds()

Abstract Returns the time bounds for a list of packets.

Syntax Ace_Tool.get_time_bounds(pkt_list)

Argument	Type	Description
pkt_list	list	The list of packets to consider.

Return Type tuple — A (min, max) pair representing the minimum and maximum times for the list of packets provided.

Details The minimum time is the earliest send time for the packets in the list. The maximum time is the latest receive time for the packets.

Component Class

The Component class is a base class for the classes that represent components of an application.

Parent Class

__builtin__.object

Subclasses

- Connection Class
- Message Class
- Network_Packet Class
- Protocol_Info Class
- Tier_Pair Class
- Transaction Class

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

Connection Class

The Connection class provides access to information about the connections in a task.

The *__iter__* method is defined for this class to be equivalent to calling *connection.get_network_packets(hide_excluded = True)*.

The method resolution order for this class is

1) Connection
2) Component
3) __builtin__.object

Parent Class

Component Class

Methods

- *get_dest_hostname()*
- *get_dest_port()*
- *get_message()*
- *get_messages()*
- *get_network_packet()*
- *get_network_packets()*
- *get_num_messages()*
- *get_num_network_packets()*
- *get_protocol_name()*
- *get_src_hostname()*
- *get_src_port()*
- *get_summary()*
- *get_tier_pair()*
- *is_excluded()*
- *is_included()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

get_dest_hostname()

Abstract Returns the host name of the destination of this connection.

Class Connection Class

Syntax connection.get_dest_hostname()

Argument	Type	Description
		no arguments

Return Type string Destination host name.

Example

```
if isinstance(obj,Ace_Tool.Connection):
   connection = obj
   src_name = connection.get_src_hostname () + ":" + str(connection.get_src_port())
   dst_name = connection.get_dest_hostname ()  + ":" +
      str(connection.get_dest_port())
   return get_connection_highest_protocol_name (connection) + ":" + src_name +
      " <-> " + dst_name
```

Details None.

get_dest_port()

Abstract Returns the port of the destination of this connection.

Class Connection Class

Syntax connection.get_dest_port()

Argument	Type	Description
		no arguments

Return Type int Destination port.

Example

```
if isinstance(obj,Ace_Tool.Connection):
   connection = obj
   src_name = connection.get_src_hostname () + ":" + str(connection.get_src_port())
   dst_name = connection.get_dest_hostname ()  + ":" +
      str(connection.get_dest_port())
   return get_connection_highest_protocol_name (connection) + ":" + src_name +
      " <-> " + dst_name
```

Details None.

get_message()

Abstract Returns the message in the connection with a given index.

Class Connection Class

Syntax connection.get_message(index)

Argument	Type	Description
index	int	Index of desired message.

Return Type Message Requested message.

Example

```
if msg_index >= 0 and msg_index < connection.get_num_messages():
   msg = connection.get_message(msg_index)
```

Details *index* must be an integer from 0 and *connection.get_num_messages()* - 1, inclusive. Other values will throw an IndexError exception.

get_messages()

Abstract Returns an iterable object of specified messages in this connection.

Class Connection Class

Syntax connection.get_messages(hide_excluded)

Argument	Type	Description
hide_excluded	bool	When False (default), the returned iterator contains all messages. When True, the iterator contains only included messages.

Return Type Item_Iterator <Message> Iterable object for requested messages in connection.

Example

```
# Build a list of all messages in the task
# Start with an empty list of messages
msg_list = []
# Get the current task
task = Ace_Tool.get_current_task()
# Iterate through all messages of all connections of all tier_pairs
for tier_pair in task.get_tier_pairs():
   for connection in tier_pair.get_connections():
      for msg in connection.get_messages():
         msg_list.append(msg)
```

Details None.

get_network_packet()

Abstract Returns the network packet in the connection with a given index.

Class Connection Class

Syntax connection.get_network_packet(index)

Argument	Type	Description
index	int	Index of desired packet.

Return Type Network_Packet Requested packet.

Example

```
if pk_index >= 0 and pk_index < connection.get_num_network_packets():
   packet = connection.get_network_packet(pk_index)
```

Details *index* must be an integer between 0 and *connection.get_num_network_packets()* - 1, inclusive. Other values will throw an IndexError exception.

get_network_packets()

Abstract Returns an iterable object of specified network packets in this connection.

Class Connection Class

Syntax connection.get_network_packets(hide_excluded)

Argument	Type	Description
hide_excluded	bool	When False (default), the returned iterator contains all network packets. When True, the iterator contains only included packets.

Return Type Item_Iterator <Network_Packet> Iterable object for requested packets in connection.

Example

```
def get_large_packets(min_size):
   # Start with an empty list of packets
   packets = []
   # Get the current task
   task = Ace_Tool.get_current_task()
   # Iterate through all the packets of all the connections of all the tier_pairs
   for tier_pair in task.get_tier_pairs():
      for connection in tier_pair.get_connections():
         for packet in connection.get_network_packets():
            # If the packet is big enough...
            if packet.get_payload_length() >= min_size:
               # Add it to the list
               packets.append(packet)
   # Hand back the list of packets
   return packets
```

Details None.

get_num_messages()

Abstract Returns the number of messages in this connection.

Class Connection Class

Syntax connection.get_num_messages()

Argument	Type	Description
		no arguments

Return Type int Number of messages in connection.

Example

```
if msg_index >= 0 and msg_index < connection.get_num_messages():
   msg = connection.get_message(msg_index)
```

Details None.

get_num_network_packets()

Abstract Returns the number of network packets in this connection.

Class Connection Class

Syntax connection.get_num_network_packets()

Argument	Type	Description
		no arguments

Return Type int Number of packets in connection.

Example

```
total_size = 0.0
conn = current_tier.get_connection (conn_index)
for packet in conn.get_network_packets():
   total_size += packet.get_true_size()
avg_size = total_size / conn.get_num_network_packets()
```

Details None.

get_protocol_name()

Abstract Returns the name of the network layer protocol for this connection.

Class Connection Class

Syntax connection.get_protocol_name()

Argument	Type	Description
		no arguments

Return Type string Name of highest protocol.

Example

```
if isinstance(obj,Ace_Tool.Connection):
   connection = obj
   src_name = connection.get_src_hostname () + ":" + str(connection.get_src_port())
   dst_name = connection.get_dest_hostname ()  + ":" +
      str(connection.get_dest_port())
   return connection.get_protocol_name () + ": " + src_name + " <-> " + dst_name
```

Details None.

get_src_hostname()

Abstract Returns the host name of the source of this connection.

Class Connection Class

Syntax connection.get_src_hostname()

Argument	Type	Description
		no arguments

Return Type string Source host name.

Example

```
if isinstance(obj,Ace_Tool.Connection):
   connection = obj
   src_name = connection.get_src_hostname () + ":" + str(connection.get_src_port())
   dst_name = connection.get_dest_hostname ()  + ":" +
      str(connection.get_dest_port())
   return connection.get_protocol_name () + ": " + src_name + " <-> " + dst_name
```

Details None.

get_src_port()

Abstract Returns the port of the source of this connection.

Class Connection Class

Syntax connection.get_src_port()

Argument	Type	Description
		no arguments

Return Type int Source port number.

Example

```
if isinstance(obj,Ace_Tool.Connection):
   connection = obj
   src_name = connection.get_src_hostname () + ":" + str(connection.get_src_port())
   dst_name = connection.get_dest_hostname ()  + ":" +
      str(connection.get_dest_port())
   return get_connection_highest_protocol_name (connection) + ":" + src_name +
      " <-> " + dst_name
```

Details None.

get_summary()

Abstract Returns a summary string for this connection.

Class Connection Class

Syntax connection.get_summary()

Argument	Type	Description
		no arguments

Return Type string Summary string.

Example

```
def get_protocol_summary(protocol):
   # Return a list of summary strings for connections with a given protocol
   # Start with an empty list of summary strings
   summaries = []
   # Get the current task
   task = Ace_Tool.get_current_task()
   # Iterate through all the connections of all the tier_pairs
   for tier_pair in task.get_tier_pairs():
      for connection in tier_pair.get_connections():
         # If the connection contains the given protocol...
         if connection.get_protocol_name() == protocol:
            # Add it to the list
            summaries.append(connection.get_summary())
   # Hand back the list of summary strings
   return summaries
```

Details The summary string contains information about the connection, including protocol, source and destination. For example:

```
TCP: 51287<->8889 TCP D=8889 S=51287    ACK=473663773 SEQ=982072696 LEN=404 WIN=32768
```

get_tier_pair()

Abstract Returns the tier pair for this connection.

Class Connection Class

Syntax connection.get_tier_pair()

Argument	Type	Description
		no arguments

Return Type Tier_Pair Tier pair for the connection.

Details None.

is_excluded()

Abstract Tests whether the connection is currently excluded.

Class Connection Class

Syntax connection.is_excluded()

Argument	Type	Description
		no arguments

Return Type bool True if the connection is excluded; False otherwise.

Details Connections can be excluded from a task in various ways:

- from the GUI (for example, with the Exclude Others operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Excluded connections are not deleted, but are removed from consideration for GUI operations. In scripts, excluded connections can be considered or not, as specified by the *hide_excluded* argument of some methods.

is_included()

Abstract Tests whether the connection is currently included.

Class Connection Class

Syntax connection.is_included()

Argument	Type	Description
		no arguments

Return Type bool True if the connection is included; False otherwise.

Details By default, all connections are included in a task. Specific connections can be excluded in various ways:

- from the GUI (for example, with the Exclude Selected Items operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Similarly, there are various ways to include connections that have been excluded:

- from the GUI (for example, with the Include Selected Items operation)
- from scripts (via the include() and include_all() methods)

Excluded connections are not deleted, but are removed from consideration by GUI operations. In scripts, excluded connections can be considered or not, as specified by the *hide_excluded* argument of some methods.

Dependency Class

The Dependency class provides access to information about the dependencies in a task.

The *__cmp__* method is defined for this class, allowing dependencies to be tested for equality.

Parent Class

__builtin__.object

Methods

- *get_application_delay()*
- *get_dependent_message()*
- *get_network_delay()*
- *get_parent_message()*
- *get_parent_network_packet()*
- *get_type()*
- *get_user_delay()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

get_application_delay()

Abstract Returns the length of the application processing delay for this dependency.

Class Dependency Class

Syntax dependency.get_application_delay()

Argument	Type	Description
		no arguments

Return Type float Application processing delay, in seconds.

Details None.

get_dependent_message()

Abstract Returns the dependent message of this object.

Class Dependency Class

Syntax dependency.get_dependent_message()

Argument	Type	Description
		no arguments

Return Type Message Dependent message.

Details None.

get_network_delay()

Abstract Returns the length of the network delay for this dependency.

Class Dependency Class

Syntax dependency.get_network_delay()

Argument	Type	Description
		no arguments

Return Type float Network delay, in seconds.

Details None.

get_parent_message()

Abstract Returns the parent message of this dependency.

Class Dependency Class

Syntax dependency.get_parent_message()

Argument	Type	Description
		no arguments

Return Type Message Parent message.

Details None.

get_parent_network_packet()

Abstract Returns the network packet that triggers this dependency.

Class Dependency Class

Syntax dependency.get_parent_network_packet()

Argument	Type	Description
		no arguments

Return Type Network_Packet — Triggering packet; “None” if the object is dependent only on the parent message, in which case the network packet will be contained in the message returned by *get_parent_message()*.

Details None.

get_type()

Abstract Returns the dependency type.

Class Dependency Class

Syntax dependency.get_type()

Argument	Type	Description
		no arguments

Return Type int One of the constants defined in the Depends_On Class.

Details None.

get_user_delay()

Abstract Returns the length of the “user think time” delay for this dependency.

Class Dependency Class

Syntax dependency.get_user_delay()

Argument	Type	Description
		no arguments

Return Type float User think time delay, in seconds.

Details None.

Depends_On Class

The Depends_On class is an enumerator class. It defines the dependency type values returned by *dependency.get_type()*.

Table 17-3 Depends_On Constants

Variable	Type	Value
FIRST	int	2
MESSAGE_ARRIVAL	int	1
MESSAGE_SEND	int	0

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

Draw_Style Class

The Draw_Style class is an enumerator class. It defines draw style options for graphs.

Table 17-4 Draw_Style Constants

Variable	Type	Value
BAR	int	3
BAR_CHART	int	4
CLUSTERED_BAR_CHART	int	12
DISCRETE	int	1
LINEAR	int	0
LINEAR_AREA	int	7
LINEAR_SYMBOL	int	6
MULTICOLOR_BAR_CHART	int	10
PIE_CHART	int	11
SAMPLE_HOLD	int	11
SPLINE	int	2
SQUARE_WAVE	int	5
STACKED_BAR_CHART	int	9

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

Message Class

The Message class provides access to information about messages.

The *__cmp__* method is defined for this class, allowing messages to be tested for equality.

The *__iter__* method is defined for this class to be equivalent to calling *message.get_network_packets(hide_excluded = True)*.

The method resolution order for this class is

1) Message
2) Component
3) __builtin__.object

Parent Class

Component Class

Methods

- *add_label()*
- *clear_labels()*
- *get_connection()*
- *get_dependent()*
- *get_dependents()*
- *get_dest_hostname()*
- *get_labels()*
- *get_message_number()*
- *get_network_packet()*
- *get_network_packets()*
- *get_num_dependents()*
- *get_num_network_packets()*
- *get_parent_dependency()*
- *get_src_hostname()*
- *get_transaction()*

- *is_excluded()*
- *is_included()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

add_label()

Abstract Adds a label to this message with given rollup, short, and long text strings.

Class Message Class

Syntax message.add_label(rollup_text, short_text, long_text)

Argument	Type	Description
rollup_text	string	Text displayed in the visualization bubble when many messages overlap, along with the number of messages (for example, “10x: <rollup_text>”).
short_text	string	Text displayed in the visualization bubble for a single message.
long_text	string	Text displayed in the tooltip when the cursor hovers over a visualization bubble.

Return Type No return value.

Details Call *task.redraw()* to display these changes in the UI.

clear_labels()

Abstract Clears all labels that have been added to this message, including any created by a visualization.

Class Message Class

Syntax message.clear_labels()

Argument	Type	Description
		no arguments

Return Type No return value.

Details None.

get_connection()

Abstract Returns the connection this message is part of.

Class Message Class

Syntax message.get_connection()

Argument	Type	Description
		no arguments

Return Type Connection Connection for this message.

Details None.

get_dependent()

Abstract Returns the message's child Dependency object at the given index.

Class Message Class

Syntax message.get_dependent(index)

Argument	Type	Description
index	int	Index of requested child dependency.

Return Type Dependency Requested child dependency.

Details The index must be an integer between 0 and *get_num_dependents()* - 1, inclusive. If not, an IndexError exception will be raised.

get_dependents()

Abstract Returns a sequence of all child Dependency objects of this message.

Class Message Class

Syntax message.get_dependents()

Argument	Type	Description
		no arguments

Return Type Item_Iterator <Dependency> Iterable object for requested dependents of message.

Details None.

get_dest_hostname()

Abstract Returns the host name of the destination tier for this message.

Class Message Class

Syntax message.get_dest_hostname()

Argument	Type	Description
		no arguments

Return Type string Destination host name.

Details None.

get_labels()

Abstract Returns a sequence of tuples containing the labels for this message.

Class Message Class

Syntax message.get_labels()

Argument	Type	Description
		no arguments

Return Type list of tuples All labels in message.

Details Each label is returned as a tuple containing the rollup, short, and long text strings for the label.

get_message_number()

Abstract Returns the message number of this message.

Class Message Class

Syntax message.get_message_number()

Argument	Type	Description
		no arguments

Return Type int Message number.

Details None.

get_network_packet()

Abstract Returns the network packet in the message with a given index.

Class Message Class

Syntax message.get_network_packet(index)

Argument	Type	Description
index	int	Index of requested network packet.

Return Type Network_Packet Requested packet.

Details *index* must be an integer between 0 and *message.get_num_network_packets()* - 1, inclusive. Other values will throw an IndexError exception.

get_network_packets()

Abstract Returns a sequence of all network packets in this message.

Class Message Class

Syntax message.get_network_packets(hide_excluded)

Argument	Type	Description
hide_excluded	bool	When False (default), the returned iterator contains all network packets. When True, the iterator contains only included packets.

Return Type Item_Iterator <Network_Packet> Iterable object for requested packets in message.

Details Some packets (such as dataless ACKs) are not included in messages. To access all packets, use the Connection Class.

get_num_dependents()

Abstract Returns the number of application messages that are dependent on this message.

Class Message Class

Syntax message.get_num_dependents()

Argument	Type	Description
		no arguments

Return Type int Number of dependents.

Details None.

get_num_network_packets()

Abstract Returns the number of network packets in this message.

Class Message Class

Syntax message.get_num_network_packets()

Argument	Type	Description
		no arguments

Return Type int Number of packets in message.

Details None.

get_parent_dependency()

Abstract Returns the parent Dependency object for this message.

Class Message Class

Syntax message.get_parent_dependency()

Argument	Type	Description
		no arguments

Return Type Dependency — Parent Dependency object; “None” if the message is either the first application message or is acausal.

Details None.

get_src_hostname()

Abstract Returns the host name of the source tier for this message.

Class Message Class

Syntax message.get_src_hostname()

Argument	Type	Description
		no arguments

Return Type string Source host name.

Details None.

get_transaction()

Abstract Returns the transaction this message belongs to.

Class Message Class

Syntax message.get_transaction()

Argument	Type	Description
		no arguments

Return Type Transaction Transaction for this message.

Details None.

is_excluded()

Abstract Tests whether the message is currently excluded.

Class Message Class

Syntax messsage.is_excluded()

Argument	Type	Description
		no arguments

Return Type bool True if the message is excluded; False otherwise.

Details Messages can be excluded from a task in various ways:

- from the GUI (for example, with the Exclude Others operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Excluded messages are not deleted, but are removed from consideration for GUI operations. In scripts, excluded messages can be considered or not, as specified by the *hide_excluded* argument of some methods.

is_included()

Abstract Tests whether the message is currently included.

Class Message Class

Syntax message.is_included()

Argument	Type	Description
		no arguments

Return Type bool True if the message is included; False otherwise.

Details By default, all messages are included in a task. Specific messages can be excluded in various ways:

- from the GUI (for example, with the Exclude Selected Items operation or the View Results and Exclude Unmatched Dialog Box
- from scripts (via the exclude() and exclude_others() methods)

Similarly, there are various ways to include messages that have been excluded:

- from the GUI (for example, with the Include Selected Items operation)
- from scripts (via the include() and include_all() methods)

Excluded messages are not deleted, but are removed from consideration by GUI operations. In scripts, excluded messages can be considered or not, as specified by the *hide_excluded* argument of some methods.

Network_Packet Class

The Network_Packet class provides access to information about the packets (frames) in a connection.

The *__cmp__* method is defined for this class, allowing network packets to be tested for equality.

The *__iter__* method is defined for this class to be equivalent to calling *network_packet.get_layers()*.

The method resolution order for this class is

1) Network_Packet
2) Component
3) __builtin__.object

Parent Class

Component Class

Methods

- *add_label()*
- *clear_labels()*
- *get_captured_size()*
- *get_connection()*
- *get_dest_hostname()*
- *get_dest_port()*
- *get_frame_number()*
- *get_labels()*
- *get_layer()*
- *get_layers()*
- *get_message()*
- *get_num_layers()*
- *get_payload_length()*
- *get_printable_string()*

- *get_recv_time()*
- *get_send_time()*
- *get_src_hostname()*
- *get_src_port()*
- *get_trace_id()*
- *get_true_size()*
- *is_excluded()*
- *is_included()*

Properties

The Network_Packet class defines numerous properties that give access to selected information about packets. This information can be useful for making decisions about how to handle a packet. For example:

```
# Show decodes for retransmitted packets
packet_list = []
task = Ace_Tool.get_current_task()
for tp in  task.get_tier_pairs():
   for conn in tp.get_connections():
      for packet in conn.get_network_packets():
         if packet["packet.is retransmit"]:
            packet_list.append(packet)
if len(packet_list) > 0:
   task.show_protocol_decodes(packet_list)
```

Table 17-5 Properties of Network_Packet Class

Property	Type	Access	Description
packet.is dropped	bool	read	True if the packet was dropped
packet.is received but not set	bool	read	True if the packet was received, but not retransmitted
packet.is retransmit	bool	read	True if the packet was retransmitted
ip.length	int	read	IP length of the packet, in bytes
ip.checksum	int	read	IP checksum of the packet
tcp.ack	int	read	TCP ACK number of the packet
tcp.seq	int	read	TCP sequence number of the packet
tcp.is urg	bool	read	True if the TCP URG flag of the packet is set
tcp.is fin	bool	read	True if the TCP FIN flag of the packet is set
tcp.is rst	bool	read	True if the TCP RST flag of the packet is set

Table 17-5 Properties of Network_Packet Class (Continued)

Property	Type	Access	Description
tcp.is ack	bool	read	True if the TCP ACK flag of the packet is set
tcp.is psh	bool	read	True if the TCP PSH flag of the packet is set
tcp.is syn	bool	read	True if the TCP SYN flag of the packet is set
tcp.window size	int	read	TCP window size of the packet, in bytes
tcp.header length	int	read	TCP header length of the packet, in bytes
http.caching	string	read	Value of any cache information contained in the packet's HTTP header (if present). This is the same information that would appear under the "Response Caching" or "Request Caching" section of the HTTP tab in AppDoctor.
http.content length	int	read	Value of the HTTP content length header
http.content type	string	read	MIME type listed in the content type header
http.request type	string	read	HTTP request method (such as GET or POST)
http.resource name	string	read	URL of the HTTP resource
http.response code	string	read	Code returned in the HTTP response (such as "200 OK" or "404 Not Found")
iiop.message size	int	read	Message size, in bytes
iiop.message type	int	read	Type of message: • -1 = Unknown/undetermined • 0 = Request message • 1 = Reply message • 3 = Locate request message • 4 = Locate reply message
iiop.relative time	double	read	Time relative to start of trace, in seconds

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

add_label()

Abstract Adds a label to this network packet.

Class Network_Packet Class

Syntax network_packet.add_label(rollup_text, short_text, long_text)

Argument	Type	Description
rollup_text	string	Text displayed in the visualization bubble when many messages overlap, along with the number of messages (for example, “10x: <rollup_text>”).
short_text	string	Text displayed in the visualization bubble for a single message.
long_text	string	Text displayed in the tooltip when the cursor hovers over a visualization bubble.

Return Type No return value.

Details Call *task.redraw()* to display these changes in the UI.

clear_labels()

Abstract Clears all labels that have been added to this packet, including any created by a visualization.

Class Network_Packet Class

Syntax network_packet.clear_labels()

Argument	Type	Description
		no arguments

Return Type No return value.

Details Call *task.redraw()* to display these changes in the UI.

get_captured_size()

Abstract Returns the captured size of this network packet.

Class Network_Packet Class

Syntax network_packet.get_capture_size()

Argument	Type	Description
		no arguments

Return Type int Captured size of packet, in bytes.

Example

```
total_size = 0.0
conn = current_tier.get_connection (conn_index)
for packet in conn.get_network_packets():
   total_size += packet.get_captured_size()
avg_size = total_size / conn.get_num_network_packets()
```

Details None.

get_connection()

Abstract Returns the connection this network packet is part of.

Class Network_Packet Class

Syntax network_packet.get_connection()

Argument	Type	Description
		no arguments

Return Type Connection Connection for this packet.

Details None.

get_dest_hostname()

Abstract Returns the host name this network packet is sent to.

Class Network_Packet Class

Syntax network_packet.get_dest_hostname()

Argument	Type	Description
		no arguments

Return Type string Destination host name.

Details None.

get_dest_port()

Abstract Returns the port this network packet is sent to.

Class Network_Packet Class

Syntax network_packet.get_dest_port()

Argument	Type	Description
		no arguments

Return Type int Destination port.

Details None.

get_frame_number()

Abstract Returns the assigned frame number of this network packet.

Class Network_Packet Class

Syntax network_packet.get_frame_number()

Argument	Type	Description
		no arguments

Return Type int Frame number of packet.

Details None.

get_labels()

Abstract Returns a sequence of tuples containing the labels for this network packet.

Class Network_Packet Class

Syntax network_packet.get_labels()

Argument	Type	Description
		no arguments

Return Type list of tuples All labels in this network packet.

Details Each label is returned as a tuple containing the rollup, short, and long text strings for the label.

get_layer()

Abstract Returns the layer in the network packet with a given index or name.

Class Network_Packet Class

Syntax network_packet.get_layer(layer)

Argument	Type	Description
layer	int or string	Index (int) or name (string) of requested layer.

Return Type Protocol_Info Requested protocol layer.

Details If *layer* is an index, it must be an integer between 0 and *network_packet.get_num_layers()* - 1, inclusive.

If *layer* is a name, it is compared (regardless of case) to each protocol name in the network packet. If a match is found, the layer is returned.

If *layer* is an invalid index or a name that does not match any layer, a PyExc_IndexError exception is thrown.

get_layers()

Abstract Returns a sequence of all protocol layers in this network packet.

Class Network_Packet Class

Syntax network_packet.get_layers()

Argument	Type	Description
		no arguments

Return Type list of Protocol_Info objects All layers in the packet.

Details None.

get_message()

Abstract Returns the message this network packet is part of.

Class Network_Packet Class

Syntax network_packet.get_message()

Argument	Type	Description
		no arguments

Return Type Message Message for this packet.

Details None.

get_num_layers()

Abstract Returns the number of layers in this network packet.

Class Network_Packet Class

Syntax network_packet.get_num_layers()

Argument	Type	Description
		no arguments

Return Type int Number of layers in the packet.

Details None.

get_payload_length()

Abstract Returns the payload length of this network packet.

Class Network_Packet Class

Syntax network_packet.get_payload_length()

Argument	Type	Description
		no arguments

Return Type int Payload length, in bytes.

Example

```
def get_large_packets(min_size):
   # Start with an empty list of packets
   packets = []
   # Get the current task
   task = Ace_Tool.get_current_task()
   # Iterate through all the packets of all the connections of all the tier_pairs
   for tier_pair in task.get_tier_pairs():
      for connection in tier_pair.get_connections():
         for packet in connection.get_network_packets():
            # If the packet is big enough...
               if packet.get_payload_length() >= min_size:
                  # Add it to the list
                  packets.append(packet)
   # Hand back the list of packets
   return packets
```

Details None.

get_printable_string()

Abstract Returns the bytes for this packet as a string, replacing unprintable characters.

Class Network_Packet Class

Syntax network_packet.get_printable_string(replacement_char, replace_cr_lf_tab)

Argument	Type	Description
replacement_char	string	Character used to replace unprintable characters (default is '.').
replace_cr_lf_tab	bool	When True (default), carriage returns, line feeds, and tabs are replaced.

Return Type string Layer contents.

Details Returns the bytes for this packet as a string, replacing unprintable characters with *replacement_char*. Common replacements are dot (.) and space ( ).

By default, carriage returns, linefeeds, and tabs are also replaced. To leave them untouched, pass *replace_cr_lf_tab* as False.

get_recv_time()

Abstract Returns the time the packet was received.

Class Network_Packet Class

Syntax network_packet.get_recv_time()

Argument	Type	Description
		no arguments

Return Type float Receive time, in seconds since the start of the task.

Details None.

get_send_time()

Abstract Returns the time the packet was sent.

Class Network_Packet Class

Syntax network_packet.get_send_time()

Argument	Type	Description
		no arguments

Return Type float Send time, in seconds since the start of the task.

Details None.

get_src_hostname()

Abstract Returns the host name this network packet is sent from.

Class Network_Packet Class

Syntax network_packet.get_src_hostname()

Argument	Type	Description
		no arguments

Return Type string Source host name.

Details None.

get_src_port()

Abstract Returns the port this network packet is sent from.

Class Network_Packet Class

Syntax network_packet.get_src_port()

Argument	Type	Description
		no arguments

Return Type int Source port.

Details None.

get_trace_id()

Abstract Returns the globally unique ID for the trace to which this network packet belongs.

Class Network_Packet Class

Syntax network_packet.get_trace_id()

Argument	Type	Description
		no arguments

Return Type int Trace ID.

Details None.

get_true_size()

Abstract Returns the true size of this network packet.

Class Network_Packet Class

Syntax network_packet.get_true_size()

Argument	Type	Description
		no arguments

Return Type int True packet size, in bytes.

Example

```
total_size = 0.0
conn = current_tier.get_connection (conn_index)
for packet in conn.get_network_packets():
   total_size += packet.get_true_size()
avg_size = total_size / conn.get_num_network_packets()
```

Details None.

is_excluded()

Abstract Tests whether the network packet is currently excluded.

Class Network_Packet Class

Syntax network_packet.is_excluded()

Argument	Type	Description
		no arguments

Return Type bool True if the network packet is excluded; False otherwise.

Details Network packets can be excluded from a task in various ways:

- from the GUI (for example, with the Exclude Others operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Excluded packets are not deleted, but are removed from consideration for GUI operations. In scripts, excluded packets can be considered or not, as specified by the *hide_excluded* argument of some methods.

is_included()

Abstract Tests whether the network packet is currently included.

Class Network_Packet Class

Syntax network_packet.is_included()

Argument	Type	Description
		no arguments

Return Type bool True if the network packet is included; False otherwise.

Details By default, all network packets are included in a task. Specific packets can be excluded in various ways:

- from the GUI (for example, with the Exclude Selected Items operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Similarly, there are various ways to include network packets that have been excluded:

- from the GUI (for example, with the Include Selected Items operation)
- from scripts (via the include() and include_all() methods)

Excluded packets are not deleted, but are removed from consideration by GUI operations. In scripts, excluded packets can be considered or not, as specified by the *hide_excluded* argument of some methods.

Pref_Type Class

The Pref_Type class is an enumerator class. It specifies data types for input parameter dialog boxes.

Table 17-6 Pref_Type Constants

Variable	Type	Value
EDIT_ENUM	int	2
ENUM	int	1
STRING	int	0

Protocol_Info Class

The Protocol_Info class provides access to protocol information about a layer of a network packet.

The *__cmp__* method is defined for this class, allowing protocols to be tested for equality.

The method resolution order for this class is

1) Protocol_Info
2) Component
3) __builtin__.object

Parent Class

Component Class

Methods

- *get_bytes()*
- *get_bytes_tuple()*
- *get_details()*
- *get_id()*
- *get_name()*
- *get_network_packet()*
- *get_offset()*
- *get_printable_string()*
- *get_string()*
- *get_summary()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

get_bytes()

Abstract Returns the bytes for this protocol layer as an array of unsigned integers.

Class Protocol_Info Class

Syntax protocol_info.get_bytes(include_higher_layers)

Argument	Type	Description
include_higher_layers	bool	When False (default), only bytes from this layer are returned. When True, bytes from this and all higher layers are returned.

Return Type array of unsigned ints Layer contents.

Details The byte range returned is from the offset of this layer up to, but not including, the offset of the following layer. If this is the last layer, the byte range is to the end of the packet.

get_bytes_tuple()

Abstract Returns the bytes for this layer as a tuple of unsigned integers.

Class Protocol_Info Class

Syntax protocol_info.get_bytes_tuple(include_higher_layers)

Argument	Type	Description
include_higher_layers	bool	When False (default), only bytes from this layer are returned. When True, bytes from this and all higher layers are returned.

Return Type tuple of unsigned ints Layer contents.

Details The byte range returned is from the offset of this layer up to, but not including, the offset of the following layer. If this is the last layer, the byte range is to the end of the packet.

get_details()

Abstract Returns a sequence of detail lines from the decoder for this layer.

Class Protocol_Info Class

Syntax protocol_info.get_details()

Argument	Type	Description
		no arguments

Return Type list of strings Decoded layer contents.

Details None.

get_id()

Abstract Returns the protocol ID for this layer.

Class Protocol_Info Class

Syntax protocol_info.get_id()

Argument	Type	Description
		no arguments

Return Type int Protocol ID.

Details You can use this method to get the ID argument for *task.get_protocol_name()*.

get_name()

Abstract Returns the protocol name for this layer.

Class Protocol_Info Class

Syntax protocol_info.get_name()

Argument	Type	Description
		no arguments

Return Type string Protocol name.

Details None.

get_network_packet()

Abstract Returns the network packet for this connection.

Class Protocol_Info Class

Syntax protocol_info.get_network_packet()

Argument	Type	Description
		no arguments

Return Type Network_Packet Packet for this connection.

Details None.

get_offset()

Abstract Returns the offset of this protocol.

Class Protocol_Info Class

Syntax protocol_info.get_offset()

Argument	Type	Description
		no arguments

Return Type int — Protocol offset, in bytes relative to the beginning of the network packet.

Details None.

get_printable_string()

Abstract Returns the bytes for this layer as a string, replacing unprintable characters.

Class Protocol_Info Class

Syntax protocol_info.get_printable_string(replacement_char, replace_cr_lf_tab, include_higher_layers)

Argument	Type	Description
replacement_char	string	Character used to replace unprintable characters (default is '.').
replace_cr_lf_tab	bool	When True (default), carriage returns, line feeds and tabs are replaced.
include_higher_layers	bool	When False (default), only bytes from this layer are returned. When True, bytes from this and all higher layers are returned.

Return Type string Layer contents.

Details Returns the bytes for this layer as a string, replacing unprintable characters with *replacement_char*. Common replacements are dot (.) and space ( ).

By default, carriage returns, linefeeds, and tabs are also replaced. To leave them untouched, pass *replace_cr_lf_tab* as False.

The byte range that will be converted to a string is from the offset of this layer up to, but not including, the offset of the following layer. If this is the last layer, the byte range is to the end of the packet.

get_string()

Abstract Returns the bytes for this layer as a string.

Class Protocol_Info Class

Syntax protocol_info.get_string(include_higher_layers)

Argument	Type	Description
include_higher_layers	bool	When False (default), only bytes from this layer are returned. When True, bytes from this and all higher layers are returned.

Return Type string Layer contents.

Details The byte range that will be converted to a string is from the offset of this layer up to, but not including, the offset of the following layer. If this is the last layer, the byte range is to the end of the packet.

get_summary()

Abstract Returns the summary from the decoder for this layer.

Class Protocol_Info Class

Syntax protocol_info.get_summary()

Argument	Type	Description
		no arguments

Return Type string Summary string.

Example

```
def get_layer_protocol_summary(protocol):
   # Return a list of summary strings for layers with a given protocol
   # Start with an empty list of summary strings
   summaries = []
   # Get the current task
   task = Ace_Tool.get_current_task()
   # Iterate through all layers of all packets of all connections of all tier_pairs
   for tier_pair in task.get_tier_pairs():
      for connection in tier_pair.get_connections():
         for packet in connection.get_network_protocols():
            for protocol in packet.get_layers:
               # If the layer uses the given protocol...
               if protocol.get_name() == protocol:
                  # Add it to the list
                  summaries.append(protocol.get_summary())
   # Hand back the list of summary strings
   return summaries
```

Details The summary string contains information about the protocol, including name, source, and destination. For example:

```
TCP D=8889 S=51287     ACK=473663773 SEQ=982072696 LEN=404 WIN=32768
```

Task Class

The Task class provides access to and control of tasks.

You can save arbitrary state information onto a Task object by defining new properties. This can be used, for example, to create advanced multi-step workflows in which some data needs to be saved for later use. To define a new property, use code such as

```
task = Ace_Tool.get_current_task()
task["new_property"] = <value>
```

The *__setitem__*, *__getitem__*, and *__delitem__* methods are defined for this class to implement this state-saving capability.

The *__cmp__* method is defined for this class, allowing network packets to be tested for equality.

The *__iter__* method is defined for this class to be equivalent to calling *task.get_tier_pairs(hide_excluded = True)*.

Parent Class

__builtin__.object

Methods

- *clear_labels()*
- *delete()*
- *delete_others()*
- *exclude()*
- *exclude_others()*
- *get_model_name()*
- *get_num_tier_pairs()*
- *get_num_transactions()*
- *get_protocol_name()*
- *get_root_messages()*
- *get_selected_messages()*
- *get_selected_network_packets()*
- *get_state_dict()*

- *get_tier_pair()*
- *get_tier_pairs()*
- *get_transaction()*
- *get_transactions()*
- *get_visualization_manager()*
- *has_decode_version()*
- *include()*
- *include_all()*
- *is_binary_available()*
- *redraw()*
- *get_visible_time_range()*
- *show_dec()*
- *show_protocol_decodes()*
- *show_tier_pair_circle()*
- *show_tree_view()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

clear_labels()

Abstract Clears all labels that have been added to messages or network packets, including any created by the current visualization.

Class Task Class

Syntax task.clear_labels()

Argument	Type	Description
		no arguments

Return Type No return value.

Details None.

delete()

Abstract Permanently deletes components in a given list.

Class Task Class

Syntax task.delete(delete_list)

Argument	Type	Description
delete_list	List of Component objects	List of components to be deleted.

Return Type No return value.

Details *delete_list* can be of mixed component sub-types (such as Transactions, Messages, and Network Packets). Because a GUI recalculation is performed after each call to this method, you should ideally build the full list and call this method once.

delete_others()

Abstract Permanently deletes components not in a given list.

Class Task Class

Syntax task.delete_others(keep_list)

Argument	Type	Description
keep_list	List of Component objects	List of components to be retained.

Return Type No return value.

Details *keep_list* can be of mixed component sub-types (such as Transactions, Messages, and Network Packets). Because a GUI recalculation is performed after each call to this method, you should ideally build the full list and call this method once.

exclude()

Abstract Excludes components in a given list.

Class Task Class

Syntax task.exclude(exclude_list)

Argument	Type	Description
exclude_list	List of Component objects	List of components to be excluded.

Return Type No return value.

Example

```
large_size = 512

# get_large_packets(min_size)
# User-defined function that returns a list of packets > min_size

large_packets = get_large_packets(large_size)
if len(large_packets) > 0:
   # Get the current task and give it the list to exclude
   Ace_Tool.get_current_task().exclude(large_packets)
```

Details Excluded connections are not deleted, but are removed from consideration by GUI operations. In scripts, excluded connections can be considered or not, as specified by the *hide_excluded* argument of some methods.

exclude_list can be of mixed component sub-types (such as Transaction, Message, and Network_Packet objects). Because a GUI recalculation is performed after each call to this method, it is best to build the full *exclude_list* and call this method once.

exclude_others()

Abstract Excludes all components not in a given list.

Class Task Class

Syntax task.exclude_others(consider_list)

Argument	Type	Description
consider_list	List of Component objects	List of components not to be excluded.

Return Type No return value.

Details Excluded connections are not deleted, but are removed from consideration by GUI operations. In scripts, excluded connections can be considered or not, as specified by the *hide_excluded* argument of some methods.

consider_list can be of mixed component sub-types (such as Transaction, Message, and Network_Packet objects). Because a GUI recalculation is performed after each call to this method, it is best to build the full *consider_list* and call this method once.

get_model_name()

Abstract Returns the AppTransaction Xpert model name.

Class Task Class

Syntax task.get_model_name()

Argument	Type	Description
		no arguments

Return Type string AppTransaction Xpert model name.

Details None.

get_num_tier_pairs()

Abstract Returns the number of tier pairs in this task.

Class Task Class

Syntax task.get_num_tier_pairs()

Argument	Type	Description
		no arguments

Return Type int Number of tier pairs.

Details None.

get_num_transactions()

Abstract Returns the number of transactions in this task.

Class Task Class

Syntax task.get_num_transactions()

Argument	Type	Description
		no arguments

Return Type int Number of transactions.

Details None.

get_protocol_name()

Abstract Returns the protocol name associated with the given AppTransaction Xpert protocol ID.

Class Task Class

Syntax task.get_protocol_name(id)

Argument	Type	Description
id	int	AppTransaction Xpert ID for protocol of interest.

Return Type string Protocol name.

Details Use *protocol_info.get_id()* to get the *id* argument for this method.

get_root_messages()

Abstract Returns a sequence of all messages that have no parent dependency.

Class Task Class

Syntax task.get_root_messages()

Argument	Type	Description
		no arguments

Return Type list of Message objects Root messages.

Details None.

get_selected_messages()

Abstract Returns all messages selected in the Data Exchange Chart.

Class Task Class

Syntax task.get_selected_messages()

Argument	Type	Description
		no arguments

Return Type list of Message objects Selected messages.

Details If network packets are selected in the Network Chart, the corresponding application messages (if any) are returned.

get_selected_network_packets()

Abstract Returns all network packets selected in the Data Exchange Chart.

Class Task Class

Syntax task.get_selected_network_packets()

Argument	Type	Description
		no arguments

Return Type list of Network_Packet objects Selected packets.

Details If items are selected in the Application Message Chart, all included network packets for those messages are returned.

get_state_dict()

Abstract Returns a state dictionary that contains user-specific data.

Class Task Class

Syntax task.get_state_dict()

Argument	Type	Description
		no arguments

Return Type Python object Requested state dictionary.

Details None.

get_tier_pair()

Abstract Returns the tier pair in the task with a given index.

Class Task Class

Syntax task.get_tier_pair(index)

Argument	Type	Description
index	int	Index of desired tier pair.

Return Type Tier_Pair Requested tier pair.

Details *index* must be an integer between 0 and *task.get_num_tier_pairs()* - 1, inclusive. Other values will throw an IndexError exception.

get_tier_pairs()

Abstract Returns a sequence of all tier pairs in this task.

Class Task Class

Syntax task.get_tier_pairs(hide_excluded)

Argument	Type	Description
hide_excluded	bool	When False (default), the returned iterator contains all tier pairs. When True, the iterator contains only included tier pairs.

Return Type Item_Iterator <Tier_Pair> Iterable object for requested tier pairs in task.

Example

```
def get_large_packets(min_size):
   # Start with an empty list of packets
   packets = []
   # Get the current task
   task = Ace_Tool.get_current_task()
   # Iterate through all the packets of all the connections of all the tier_pairs
   for tier_pair in task.get_tier_pairs():
      for connection in tier_pair.get_connections():
         for packet in connection.get_network_packets():
            # If the packet is big enough...
            if packet.get_payload_length() >= min_size:
               # Add it to the list
               packets.append(packet)
   # Hand back the list of packets
   return packets
```

Details None.

get_transaction()

Abstract Returns the transaction in the task with a given index.

Class Task Class

Syntax task.get_transaction(index)

Argument	Type	Description
index	int	Index of desired transaction.

Return Type Transaction Requested transaction.

Details *index* must be an integer between 0 and *task.get_num_transactions()* - 1, inclusive. Other values will throw an IndexError exception.

get_transactions()

Abstract Returns a sequence of all transactions in this task.

Class Task Class

Syntax task.get_transactions(hide_excluded)

Argument	Type	Description
hide_excluded	bool	When False (default), the returned iterator contains all transactions. When True, the iterator contains only included transactions.

Return Type Item_Iterator <Transaction> Iterable object for requested transactions in task.

Details None.

get_visualization_manager()

Abstract Returns the visualization manager object that can be used to set or get the task's UI visualization.

Class Task Class

Syntax task.get_visualization_manager()

Argument	Type	Description
		no arguments

Return Type Visualization_Manager Visualization manager object.

Details None.

has_decode_version()

Abstract Determines whether a decode exists for a particular version string.

Class Task Class

Syntax task.has_decode_version(version)

Argument	Type	Description
version	string	Version to check for a decode.

Return Type bool True if a decode exists; False otherwise.

Details None.

include()

Abstract Includes items from a list of components.

Class Task Class

Syntax task.include(include_list)

Argument	Type	Description
include_list	List of Component objects	List of components to be included.

Return Type No return value.

Example

```
large_size = 512

# get_large_packets(min_size)
# User-defined function that returns a list of packets > min_size

large_packets = get_large_packets(large_size)
if len(large_packets) > 0:
   # Get the current task and give it the list to include
   Ace_Tool.get_current_task().include(large_packets)
```

Details This method includes the components listed in *include_list*. To include all components and sub-components of the task, use *task.include_all()*.

include_list can be of mixed component sub-types (such as Transaction, Message, and Network_Packet objects). Because a GUI recalculation is performed after each call to this method, it is best to build the full *include_list* and call this method once.

include_all()

Abstract Includes all items and sub-components.

Class Task Class

Syntax task.include_all()

Argument	Type	Description
		no arguments

Return Type No return value.

Details A GUI recalculation is performed after each call to this method.

is_binary_available()

Abstract Determines whether the binary data for this trace is available.

Class Task Class

Syntax task.is_binary_available()

Argument	Type	Description
		no arguments

Return Type bool True if the binary data is available; False otherwise.

Details None.

redraw()

Abstract Redraws the AppTransaction Xpert window so that changes made by script calls are displayed.

Class Task Class

Syntax task.redraw()

Argument	Type	Description
		no arguments

Return Type No return value.

Details This operation might take a while to complete, so it should be called once after all updates have been made.

get_visible_time_range()

Abstract Sets the visible time range of the Data Exchange Chart.

Class Task Class

Syntax task.get_visible_time_range(min_time, max_time)

Argument	Type	Description
min_time	int	Start of visible time range, in seconds since start of task.
max_time	int	End of visible time range, in seconds since start of task.

Return Type No return value.

Details The number of tiers visible are not affected.

show_dec()

Abstract Shows the Data Exchange Chart in the AppTransaction Xpert window.

Class Task Class

Syntax task.show_dec()

Argument	Type	Description
		no arguments

Return Type No return value.

Details None.

show_protocol_decodes()

Abstract Shows the protocol decodes for a list of components.

Class Task Class

Syntax task.show_protocol_decodes(show_list)

Argument	Type	Description
show_list	List of Component objects	List of components for which protocol decodes should be shown.

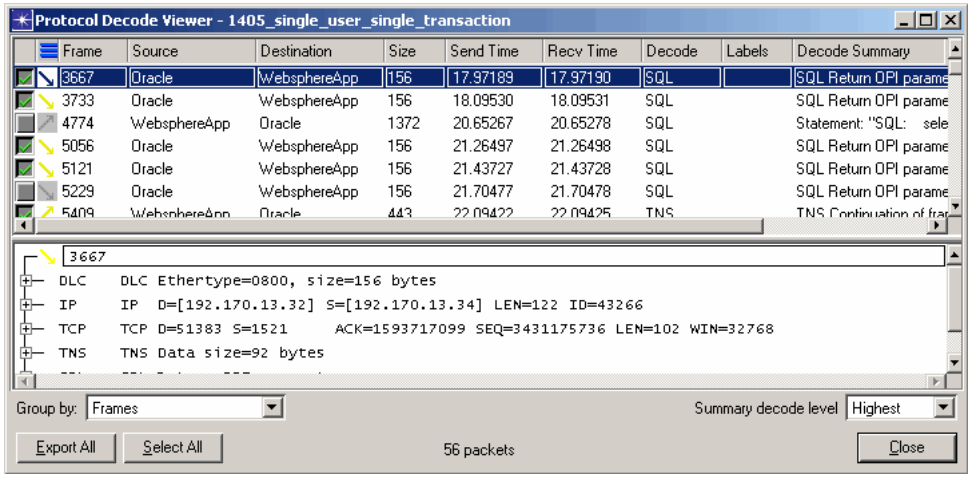
Return Type No return value.

Example

```
def show_decodes_for(attrib, value):
   """Show Protocol Decodes for all network packets
      whose attribute has a certain value"""
   task = Ace_Tool.get_current_task()
   npks = []
   for tp in task.get_tier_pairs():
      for conn in tp.get_connections():
         for npk in conn.get_network_packets():
            if npk[attrib] == value:
               npks.append(npk)
   if len(npks) > 0:
      task.show_protocol_decodes(npks)

show_decodes_for("packet.is retransmit", 1)
```

Figure 17-9 Protocol Decode



Details *show_list* can be of mixed component sub-types (such as Transactions, Messages, and Network Packets). Because a GUI recalculation is performed after each call to this method, you should ideally build the full list and call this method once.

show_tier_pair_circle()

Abstract Shows the Tier Pair Circle in the AppTransaction Xpert window.

Class Task Class

Syntax task.show_tier_pair_circle()

Argument	Type	Description
		no arguments

Return Type No return value.

Details None.

show_tree_view()

Abstract Shows the Tree View in the AppTransaction Xpert window.

Class Task Class

Syntax task.show_tree_view()

Argument	Type	Description
		no arguments

Return Type No return value.

Details None.

Tier Class

The Tier class provides access to information about tiers.

The *__cmp__* method is defined for this class, allowing tiers to be tested for equality.

Parent Class

__builtin__.object

Methods

- *get_hostname()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

get_hostname()

Abstract Returns the host name of a tier.

Class Tier Class

Syntax tier.get_hostname()

Argument	Type	Description
		no arguments

Return Type string Host name for tier.

Example

```
tier = tier_pair.get_src_tier()
src_host = tier.get_hostname()
```

Details You can get the host name of a tier directly from a tier pair, connection, network packet, or message by using any of the *get_src_hostname()* or *get_dest_hostname()* methods.

Tier_Pair Class

The Tier_Pair class provides access to the tier pairs in a task.

The *__cmp__* method is defined for this class, allowing tier pairs to be tested for equality.

The *__iter__* method is defined for this class to be equivalent to calling *tier_pair.get_connections(hide_excluded = True)*.

The method resolution order for this class is

1) Tier_Pair
2) Component
3) __builtin__.object

Parent Class

Component Class

Methods

- *get_connection()*
- *get_connections()*
- *get_dest_hostname()*
- *get_dest_tier()*
- *get_num_connections()*
- *get_src_hostname()*
- *get_src_tier()*
- *is_excluded()*
- *is_included()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

get_connection()

Abstract Returns the connection in the tier pair with a given index.

Class Tier_Pair Class

Syntax tier_pair.get_connection(index)

Argument	Type	Description
index	int	Index of the connection of interest.

Return Type Connection Requested connection.

Details *index* must be an integer between 0 and *tier_pair.get_num_connections()* - 1, inclusive. Other values will throw an IndexError exception.

get_connections()

Abstract Returns a sequence of all connections in this tier pair.

Class Tier_Pair Class

Syntax tier_pair.get_connections(hide_excluded)

Argument	Type	Description
hide_excluded	bool	When False (default), the returned iterator contains all connections. When True, the iterator contains only included connections.

Return Type Item_Iterator <Connection> Iterable object for requested connections in tier pair.

Example

```
def get_large_packets(min_size):
   # Start with an empty list of packets
   packets = []
   # Get the current task
   task = Ace_Tool.get_current_task()
   # Iterate through all the packets of all the connections of all the tier_pairs
   for tier_pair in task.get_tier_pairs():
      for connection in tier_pair.get_connections():
         for packet in connection.get_network_packets():
            # If the packet is big enough...
            if packet.get_payload_length() >= min_size:
               # Add it to the list
               packets.append(packet)
   # Hand back the list of packets
   return packets
```

Details None.

get_dest_hostname()

Abstract Returns the host name of the destination tier.

Class Tier_Pair Class

Syntax tier_pair.get_dest_hostname()

Argument	Type	Description
		no arguments

Return Type string Destination host name.

Details None.

get_dest_tier()

Abstract Returns the destination tier of the tier pair.

Class Tier_Pair Class

Syntax tier_pair.get_dest_tier()

Argument	Type	Description
		no arguments

Return Type Tier Destination tier.

Details None.

get_num_connections()

Abstract Returns the number of connections in this tier pair.

Class Tier_Pair Class

Syntax tier_pair.get_num_connections()

Argument	Type	Description
		no arguments

Return Type int Number of connections.

Details None.

get_src_hostname()

Abstract Returns the host name of the source tier.

Class Tier_Pair Class

Syntax tier_pair.src_hostname()

Argument	Type	Description
		no arguments

Return Type string Source host name.

Details None.

get_src_tier()

Abstract Returns the source tier of the tier pair.

Class Tier_Pair Class

Syntax tier_pair.get_src_tier()

Argument	Type	Description
		no arguments

Return Type Tier Source tier.

Details None.

is_excluded()

Abstract Tests whether the tier pair is currently excluded.

Class Tier_Pair Class

Syntax tier pair.is_excluded()

Argument	Type	Description
		no arguments

Return Type bool True if the tier pair is excluded; False otherwise.

Details Tier pairs can be excluded from a task in various ways:

- from the GUI (for example, with the Exclude Others operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Excluded tier pairs are not deleted, but are removed from consideration for GUI operations. In scripts, excluded tier pairs can be considered or not, as specified by the *hide_excluded* argument of some methods.

is_included()

Abstract Tests whether the tier pair is currently included.

Class Tier_Pair Class

Syntax tier pair.is_included()

Argument	Type	Description
		no arguments

Return Type bool True if the tier pair is included; False otherwise.

Details By default, all tier pairs are included in a task. Specific tier pairs can be excluded in various ways:

- from the GUI (for example, with the Exclude Selected Items operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Similarly, there are various ways to include tier pairs that have been excluded:

- from the GUI (for example, with the Include Selected Items operation)
- from scripts (via the include() and include_all() methods)

Excluded tier pairs are not deleted, but are removed from consideration by GUI operations. In scripts, excluded tier pairs can be considered or not, as specified by the *hide_excluded* argument of some methods.

Transaction Class

The Transaction class provides access to the transactions in a task.

The *__cmp__* method is defined for this class, allowing transactions to be tested for equality.

The *__iter__* method is defined for this class to be equivalent to calling *transaction.get_messages(hide_excluded = True)*.

The method resolution order for this class is

1) Transaction
2) Component
3) __builtin__.object

Parent Class

Component Class

Methods

- *get_message()*
- *get_messages()*
- *get_num_messages()*
- *is_excluded()*
- *is_included()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

get_message()

Abstract Returns the message in the transaction with a given index.

Class Transaction Class

Syntax transaction.get_message(index)

Argument	Type	Description
index	int	Index of the message of interest.

Return Type Message Requested transaction.

Example

```
if msg_index >= 0 and msg_index < transaction.get_num_messages():
   msg = transaction.get_message(msg_index)
```

Details *index* must be an integer between 0 and *transaction.get_num_messages()* - 1, inclusive. Other values will throw an IndexError exception.

get_messages()

Abstract Returns a sequence of all messages in this transaction.

Class Transaction Class

Syntax transaction.get_messages(hide_excluded)

Argument	Type	Description
hide_excluded	bool	When False (default), the returned iterator contains all messages. When True, the iterator contains only included messages.

Return Type Item_Iterator <Message> Iterable object for requested messages in transaction.

Example

```
# Build a list of all transaction messages in the task
# Start with an empty list of messages
msg_list = []
# Get the current task
task = Ace_Tool.get_current_task()
# Iterate through all messages of all transactions
for transaction in task.get_transactions():
   for msg in transaction.get_messages():
      msg_list.append(msg)
```

Details None.

get_num_messages()

Abstract Returns the number of messages in this transaction.

Class Transaction Class

Syntax transaction.get_num_messages()

Argument	Type	Description
		no arguments

Return Type int Number of messages.

Example

```
if msg_index >= 0 and msg_index < transaction.get_num_messages():
   msg = transaction.get_message(msg_index)
```

Details None

is_excluded()

Abstract Tests whether the transaction is currently excluded.

Class Transaction Class

Syntax transaction.is_excluded()

Argument	Type	Description
		no arguments

Return Type bool True if the transaction is excluded; False otherwise.

Details Transactions can be excluded from a task in various ways:

- from the GUI (for example, with the Exclude Others operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Excluded transactions are not deleted, but are removed from consideration for GUI operations. In scripts, excluded transactions can be considered or not, as specified by the *hide_excluded* argument of some methods.

is_included()

Abstract Tests whether the transaction is currently included.

Class Transaction Class

Syntax transaction.is_included()

Argument	Type	Description
		no arguments

Return Type bool True if the transaction is included; False otherwise.

Details By default, all transactions are included in a task. Specific transactions can be excluded in various ways:

- from the GUI (for example, with the Exclude Selected Items operation or the View Results and Exclude Unmatched Dialog Box)
- from scripts (via the exclude() and exclude_others() methods)

Similarly, there are various ways to include transactions that have been excluded:

- from the GUI (for example, with the Include Selected Items operation)
- from scripts (via the include() and include_all() methods)

Excluded transactions are not deleted, but are removed from consideration by GUI operations. In scripts, excluded transactions can be considered or not, as specified by the *hide_excluded* argument of some methods.

Visualization_Manager Class

The Visualization_Manager class provides access to the visualizations in a task.

Parent Class

__builtin__.object

Methods

- *get_current_visualization()*
- *get_visualizations()*
- *set_visualization()*

Related Topics

- *Ace_Tool Classes*
- *Ace_Tool Functions*

get_current_visualization()

Abstract Returns the name of the visualization currently used in the UI.

Class Visualization_Manager Class

Syntax visualization_manager.get_current_visualization()

Argument	Type	Description
		no arguments

Return Type string Visualization name.

Details None.

get_visualizations()

Abstract Returns a sequence of available visualizations for this task.

Class Visualization_Manager Class

Syntax visualization_manager.get_visualizations()

Argument	Type	Description
		no arguments

Return Type list of strings — Names of all available visualizations.

Details None.

set_visualization()

Abstract Sets a visualization in the UI.

Class Visualization_Manager Class

Syntax visualization_manager.set_visualization(viz_name)

Argument	Type	Description
viz_name	string	Name of visualization to use.

Return Type No return value.

Details A ValueError is raised if an invalid name is specified.

Ace_Tool_Contributed Module

The Ace_Tool_Contributed module supplements the scripting API with some additional classes and functions that you can use for a variety of common tasks, such as

- handling input and output
- obtaining a sorted list of packets
- writing a custom visualization

Details about some of the classes and functions provided for handling input and output appear in these sections:

- *create_output_table_dialog()*
- *show_confirm()*
- DetailDialog Class
- InputTable Class
- SimpleInput Class

For information about all classes and functions in the Ace_Tool_Contributed module, execute the following commands in the scripting console:

```
import Ace_Tool_Contributed
help (Ace_Tool_Contributed)
```

create_output_table_dialog()

Abstract Opens a dialog box to display tabular data.

Syntax Ace_Tool_Contributed.create_output_table_dialog(row_list)

Argument	Type	Description
row_list	list of tuples	Table data to be displayed in the dialog box (see Details).

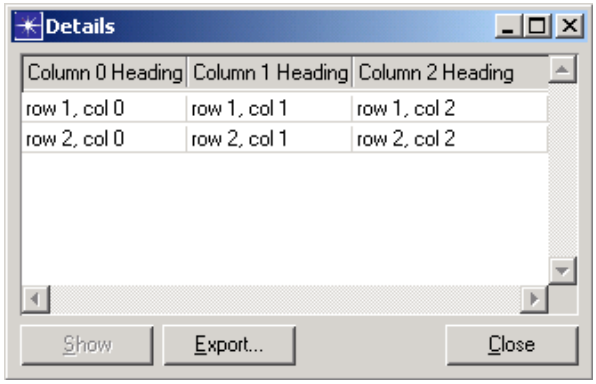
Return Type No return value.

Example

```
import Ace_Tool_Contributed

row0 = ("Column 0 Heading", "Column 1 Heading", "Column 2 Heading")
row1 = ("row 1, col 0", "row 1, col 1", "row 1, col 2")
row2 = ("row 2, col 0", "row 2, col 1", "row 2, col 2")
Ace_Tool_Contributed.create_output_table_dialog([row0, row1, row2])
```

Figure 17-10 Example Output Dialog Box for Table Data



Details This function creates an instance of a DetailDialog object and displays it. The first tuple in *row_list* specifies column headings and each additional tuple defines values for a row. You must define at least two rows (one for column headings and one for data).

show_confirm()

Abstract Opens a dialog box to display a text message.

Syntax Ace_Tool_Contributed.show_confirm(title, message)

Argument	Type	Description
title	string	Title for dialog box.
message	string	Text for dialog box.

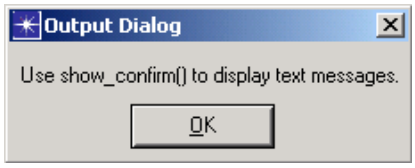
Return Type No return value.

Example

```
import Ace_Tool_Contributed

Ace_Tool_Contributed.show_confirm("Output Dialog", "Use show_confirm() to display
text messages.")
```

Figure 17-11 Example Output Dialog Box for Text Messages

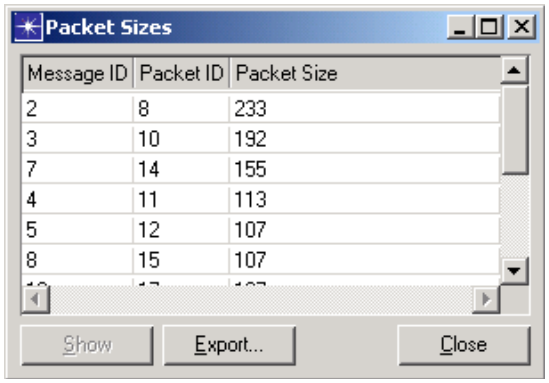


Details None.

DetailDialog Class

The DetailDialog class provides methods for building a dialog box that displays table-based data.

Figure 17-12 Example DetailDialog Dialog Box



The Show button can display additional information about a component in the task that is linked to a selected row (see the *append_row()* method for details).

The Export… button lets you save the table contents in a comma-separated (CSV) text file.

Parent Class

__builtin__.object

Methods

- *append_row()*
- *get_column_headers()*
- *get_formatted_data()*
- *get_raw_data()*
- *get_title()*
- *set_column_formatters()*
- *set_column_headers()*
- *set_title()*
- *show()*
- *sort()*

Related Topics

- *Ace_Tool_Contributed Module*

append_row()

Abstract Adds a row of data to the table of the DetailDialog object.

Class DetailDialog Class

Syntax detaildialog.append_row(component, row)

Argument	Type	Description
component	Component	A component (transaction, packet, etc.) in the task that should be tied to this row, or None if there is no callback to the row data.
row	tuple	Tuple containing data for the row. This tuple must be the same size as the headers tuple passed to detaildialog.*set_column_headers()*.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Get the current task
task = Ace_Tool.get_current_task()

# Set up the output dialog
dbox = Ace_Tool_Contributed.DetailDialog(task)
dbox.set_title("Packet Sizes")
dbox.set_column_headers(("Message ID", "Packet ID", "Packet Size"))

# Iterate through included messages of included transactions
for transaction in task.get_transactions(True):    # get included transactions
   for msg in transaction:                         # get included messages
      msg_num = msg.get_message_number()
      for packet in msg.get_network_packets():     # get all packets
         pkt_id = packet.get_frame_number()
         pkt_size = packet.get_true_size()
         dbox.append_row(packet, (msg_num, pkt_id, pkt_size))

# Sort by descending size and display
dbox.sort(2)
dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details You must specify column headers with detaildialog.*set_column_headers()* before appending data to a table.

If the *component* argument is specified, selecting a row in the output dialog box and clicking the Show button will open a Protocol Decode Viewer for the corresponding component.

get_column_headers()

Abstract Gets the column headers associated with the DetailDialog object.

Class DetailDialog Class

Syntax detaildialog.get_column_headers()

Argument	Type	Description
		no arguments

Return Type tuple Column headers.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

dbox = Ace_Tool_Contributed.DetailDialog(None)
dbox.set_column_headers(("Message ID", "Packet ID", "Packet Size"))
print dbox.get_column_headers()
```

Print output:

```
('Message ID', 'Packet ID', 'Packet Size')
```

Details None.

get_formatted_data()

Abstract Gets the table data from the DetailDialog object in formatted form.

Class DetailDialog Class

Syntax detaildialog.get_formatted_data()

Argument	Type	Description
		no arguments

Return Type list of tuples Formatted data.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

def percentFormat(data):
   # Multiplies the data point by 100 and formats to 2 decimal places.
   return str("%.2f" % (data * 100) + "%")

dbox = Ace_Tool_Contributed.DetailDialog(None)
dbox.set_column_headers(("Raw Data", "Formatted Data"))
dbox.set_column_formatters((None, percentFormat))
dbox.append_row(None, (0.003, 0.003))
dbox.append_row(None, (0.01, 0.01))
dbox.append_row(None, (0.123, 0.123))
dbox.append_row(None, (0.998765, 0.998765))
print dbox.get_formatted_data()
```

Print output:

```
[(None, (0.0030000000000000001, '0.30%')), (None, (0.01, '1.00%')), (None, (0.123,
'12.30%')), (None, (0.99876500000000001, '99.88%'))]
```

Details If no formatters have been defined with detaildialog.*set_column_formatters()*, this method returns the raw data.

get_raw_data()

Abstract Gets the table data from the DetailDialog object in raw form.

Class DetailDialog Class

Syntax detaildialog.get_raw_data()

Argument	Type	Description
		no arguments

Return Type list of tuples Raw data.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

def percentFormat(data):
   # Multiplies the data point by 100 and formats to 2 decimal places.
   return str("%.2f" % (data * 100) + "%")

dbox = Ace_Tool_Contributed.DetailDialog(None)
dbox.set_title("Raw vs. Formatted Data")
dbox.set_column_headers(("Raw Data", "Formatted Data"))
dbox.set_column_formatters((None, percentFormat))
dbox.append_row(None, (0.003, 0.003))
dbox.append_row(None, (0.01, 0.01))
dbox.append_row(None, (0.123, 0.123))
dbox.append_row(None, (0.998765, 0.998765))
print dbox.get_raw_data()
```

Print output:

```
[(None, (0.0030000000000000001, 0.0030000000000000001)), (None, (0.01, 0.01)),
(None, (0.123, 0.123)), (None, (0.99876500000000001, 0.99876500000000001))]
```

Details “Raw” data is the data given to detaildialog.*append_row()* without any formatting applied.

get_title()

Abstract Gets the title associated with the DetailDialog object.

Class DetailDialog Class

Syntax detaildialog.get_title()

Argument	Type	Description
		no arguments

Return Type string Title.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

dbox = Ace_Tool_Contributed.DetailDialog(None)
dbox.set_title(("Packet Sizes"))
print dbox.get_title()
```

Print output:

```
Packet Sizes
```

Details None.

set_column_formatters()

Abstract Sets callable routines to format the table data in the DetailDialog object.

Class DetailDialog Class

Syntax detaildialog.set_column_formatters(formatters)

Argument	Type	Description
formatters	tuple	List of callable format procedures for each column of the table. For columns that do not have a formatter, use None.

Return Type No return value.

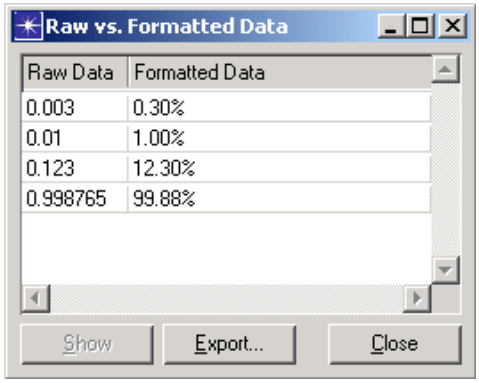
Example

```
import Ace_Tool
import Ace_Tool_Contributed

def percentFormat(data):
   # Multiplies the data point by 100 and formats to 2 decimal places.
   return str("%.2f" % (data * 100) + "%")

dbox = Ace_Tool_Contributed.DetailDialog(None)
dbox.set_title("Raw vs. Formatted Data")
dbox.set_column_headers(("Raw Data", "Formatted Data"))
dbox.set_column_formatters((None, percentFormat))
dbox.append_row(None, (0.003, 0.003))
dbox.append_row(None, (0.01, 0.01))
dbox.append_row(None, (0.123, 0.123))
dbox.append_row(None, (0.998765, 0.998765))
dbox.show()
```

Figure 17-13 Result of Column Formatting Example



Details Formatting is performed only on data appended to the table after the formatters are specified. Each formatter must be a callable that takes the column data as the argument.

set_column_headers()

Abstract Sets the column headers for the DetailDialog object.

Class DetailDialog Class

Syntax detaildialog.set_column_headers(headers)

Argument	Type	Description
headers	tuple	List of headers for each column of the table.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Get the current task
task = Ace_Tool.get_current_task()

# Set up the output dialog
dbox = Ace_Tool_Contributed.DetailDialog(task)
dbox.set_title("Packet Sizes")
dbox.set_column_headers(("Message ID", "Packet ID", "Packet Size"))

# Iterate through included messages of included transactions
for transaction in task.get_transactions(True):    # get included transactions
   for msg in transaction:                         # get included messages
      msg_num = msg.get_message_number()
      for packet in msg.get_network_packets():     # get all packets
         pkt_id = packet.get_frame_number()
         pkt_size = packet.get_true_size()
         dbox.append_row(packet, (msg_num, pkt_id, pkt_size))

# Sort by descending size and display
dbox.sort(2)
dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details This method must be called before detaildialog.*append_row()* and can be called only once.

set_title()

Abstract Sets the title for the DetailDialog dialog box.

Class DetailDialog Class

Syntax detaildialog.set_title(title)

Argument	Type	Description
title	string	Title for the dialog box.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Get the current task
task = Ace_Tool.get_current_task()

# Set up the output dialog
dbox = Ace_Tool_Contributed.DetailDialog(task)
dbox.set_title("Packet Sizes")
dbox.set_column_headers(("Message ID", "Packet ID", "Packet Size"))

# Iterate through included messages of included transactions
for transaction in task.get_transactions(True):    # get included transactions
   for msg in transaction:                         # get included messages
      msg_num = msg.get_message_number()
      for packet in msg.get_network_packets():     # get all packets
         pkt_id = packet.get_frame_number()
         pkt_size = packet.get_true_size()
         dbox.append_row(packet, (msg_num, pkt_id, pkt_size))

# Sort by descending size and display
dbox.sort(2)
dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details None.

show()

Abstract Displays the DetailDialog object.

Class DetailDialog Class

Syntax detaildialog.show()

Argument	Type	Description
		no arguments

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Get the current task
task = Ace_Tool.get_current_task()

# Set up the output dialog
dbox = Ace_Tool_Contributed.DetailDialog(task)
dbox.set_title("Packet Sizes")
dbox.set_column_headers(("Message ID", "Packet ID", "Packet Size"))

# Iterate through included messages of included transactions
for transaction in task.get_transactions(True):    # get included transactions
   for msg in transaction:                         # get included messages
      msg_num = msg.get_message_number()
      for packet in msg.get_network_packets():     # get all packets
         pkt_id = packet.get_frame_number()
         pkt_size = packet.get_true_size()
         dbox.append_row(packet, (msg_num, pkt_id, pkt_size))

# Sort by descending size and display
dbox.sort(2)
dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details After this method has been called, the object is frozen and no additional rows can be added.

sort()

Abstract Sorts the table in the DetailDialog object based on the raw data in the specified column.

Class DetailDialog Class

Syntax detaildialog.sort(column, reverse)

Argument	Type	Description
column	int	Index of the column on which to sort.
reverse	bool	Sort order. When True (default), sorts in descending order.

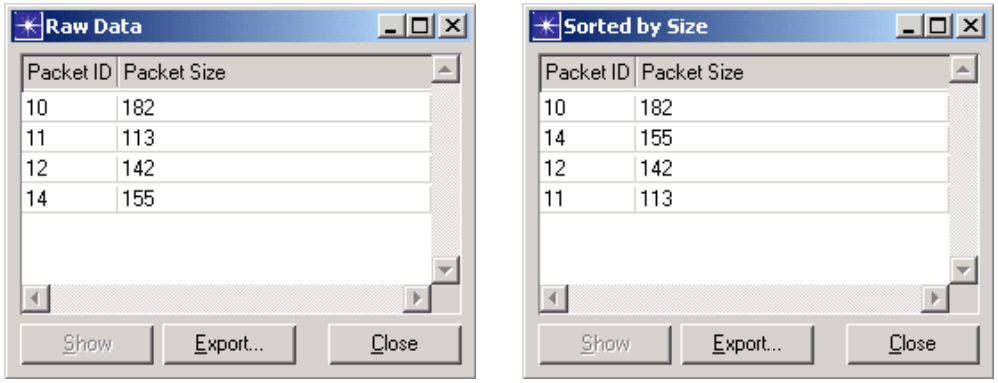
Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

dbox = Ace_Tool_Contributed.DetailDialog(None)
dbox.set_title("Raw Data")
dbox.set_column_headers(("Packet ID", "Packet Size"))
dbox.append_row(None, (10, 182))
dbox.append_row(None, (11, 113))
dbox.append_row(None, (12, 142))
dbox.append_row(None, (14, 155))
dbox.show()
# Sort in descending order by size (column 1)
dbox.sort(1)
dbox.set_title("Sorted by Size")
dbox.show()
```

Figure 17-14 Results of Sort Example

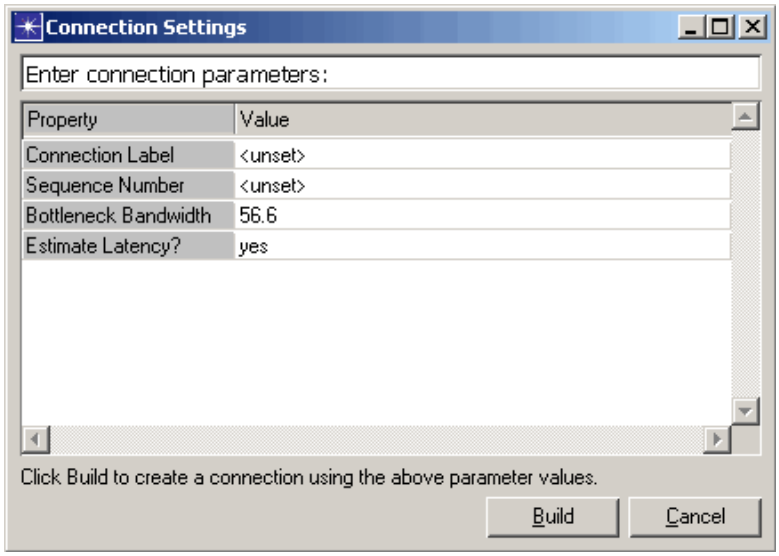


Details Once this method has been called, the object is frozen and no additional rows can be added.

InputTable Class

The InputTable class provides methods for building a table-based dialog box for getting user input.

Figure 17-15 Example InputTable Dialog Box



Parent Class

__builtin__.object

Methods

- *add_string_parameter()*
- *add_double_parameter()*
- *add_enum_parameter()*
- *add_enum_edit_parameter()*
- *set_button_label()*
- *set_title()*
- *set_bottom_text()*
- *set_top_text()*
- *show()*

Related Topics

- *Ace_Tool_Contributed Module*

add_string_parameter()

Abstract Adds a string parameter to the table of the InputTable object.

Class InputTable Class

Syntax inputtable.add_string_parameter(name, tooltip, default, out_name)

Argument	Type	Description
name	string	Parameter name (to be displayed in the Property column).
tooltip	string	Not implemented. Use "" or None.
default	string	Default value (to be displayed in the Value column). Use "" or None for no default value.
out_name	string	Name to use as the key for this parameter value in the output dictionary.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

add_double_parameter()

Abstract Adds a double parameter to the table of the InputTable object.

Class InputTable Class

Syntax inputtable.add_double_parameter(name, tooltip, default, out_name)

Argument	Type	Description
name	string	Parameter name (to be displayed in the Property column).
tooltip	string	Not implemented. Use "" or None.
default	string	Default value (to be displayed in the Value column). Use "" or None for no default value.
out_name	string	Name to use as the key for this parameter value in the output dictionary.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

add_enum_parameter()

Abstract Adds a parameter to the table of the InputTable object whose value is chosen from a given sequence of options.

Class InputTable Class

Syntax inputtable.add_enum_parameter(name, tooltip, default, options, out_name)

Argument	Type	Description
name	string	Parameter name (to be displayed in the Property column).
tooltip	string	Not implemented. Use "" or None.
default	string	Default value (to be displayed in the Value column). Must be one of the items in *options*. Use "" or None for no default value.
options	list	List of strings giving optional values for the parameter.
out_name	string	Name to use as the key for this parameter value in the output dictionary.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

add_enum_edit_parameter()

Abstract Adds a parameter to the table of the InputTable object whose value is either a string or taken from a given sequence of options.

Class InputTable Class

Syntax inputtable.add_enum_edit_parameter(name, tooltip, default, options, out_name)

Argument	Type	Description
name	string	Parameter name (to be displayed in the Property column).
tooltip	string	Not implemented. Use "" or None.
default	string	Default value (to be displayed in the Value column). Must be one of the items in *options* or “Edit...”. Use "" or None for no default value.
options	list	List of strings giving optional values for the parameter. (In the dialog box, “Edit...” is displayed automatically at the end of the options list, allowing the user to enter a custom value.)
out_name	string	Name to use as the key for this parameter value in the output dictionary.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

set_button_label()

Abstract Sets the label on the “OK” button of the InputTable object.

Class InputTable Class

Syntax inputtable.set_button_label(label)

Argument	Type	Description
label	string	Text for the “OK” button in the dialog box. If "" or None is passed, the default text “OK” is used.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

set_title()

Abstract Sets the dialog box title for the InputTable object.

Class InputTable Class

Syntax inputtable.set_title(title)

Argument	Type	Description
title	string	Text for the dialog box title. If "" or None is passed, "Specify Input" is used as the title.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

set_bottom_text()

Abstract Sets the text that appears below the parameters table in the InputTable dialog box.

Class InputTable Class

Syntax inputtable.set_bottom_text(text)

Argument	Type	Description
text	string	Bottom text for the dialog box. If "" or None is passed, “Push OK when done.” appears as the bottom text. (“OK” is replaced by the button text if *set_button_label()* has been called.)

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

set_top_text()

Abstract Sets the text that appears above the parameters table in the InputTable dialog box.

Class InputTable Class

Syntax inputtable.set_top_text(text)

Argument	Type	Description
text	string	Top text for the dialog box. If "" or None is passed, "Enter values below:" appears as the top text.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

Sample output for this example appears in Figure 17-15.

Details None.

show()

Abstract Displays the InputTable dialog box and returns the user response.

Class InputTable Class

Syntax inputtable.show()

Argument	Type	Description
		no arguments

Return Type dictionary Returns “None” if the user presses Cancel, otherwise a dictionary of parameter names and supplied values.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

# Set up dialog box
input_dbox = Ace_Tool_Contributed.InputTable()
input_dbox.set_title("Connection Settings")
input_dbox.set_top_text("Enter connection parameters:")
input_dbox.set_bottom_text("Click Build to create a connection using the above
parameter values.")
input_dbox.set_button_label("Build")

# Specify parameters
input_dbox.add_string_parameter("Connection Label", "", "<unset>", "label")
input_dbox.add_double_parameter("Sequence Number", "", "<unset>", "sequence")
input_dbox.add_enum_edit_parameter("Bottleneck Bandwidth", "", "56.6", ("28.8",
"56.6", "ISDN"), "bandwidth")
input_dbox.add_enum_parameter("Estimate Latency?", "", "yes", ("yes", "no"),
"latency")

# Get results
settings = input_dbox.show()
```

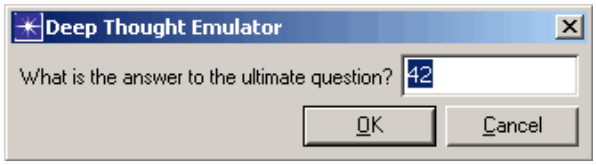
Sample output for this example appears in Figure 17-15.

Details This call is blocking.

SimpleInput Class

The SimpleInput class provides methods for building a dialog box with a single input field.

Figure 17-16 Example SimpleInput Dialog Box



Parent Class

__builtin__.object

Methods

- *set_title()*
- *set_label()*
- *set_default_value()*
- *show()*

Related Topics

- *Ace_Tool_Contributed Module*

set_title()

Abstract Sets the title for the SimpleInput dialog box.

Class SimpleInput Class

Syntax simpleinput.set_title(title)

Argument	Type	Description
title	string	Title for the dialog box. If "" is passed, the default title ("Specify Input") is used. If None is passed, "Enter Value" is used.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

dbox = Ace_Tool_Contributed.SimpleInput()
dbox.set_title("Deep Thought Emulator")
dbox.set_label("What is the answer to the ultimate question?")
dbox.set_default_value("42")
answer = dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details None.

set_label()

Abstract Sets the label for the SimpleInput dialog box.

Class SimpleInput Class

Syntax simpleinput.set_label(label)

Argument	Type	Description
label	string	Label for the dialog box. For no label, pass "".

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

dbox = Ace_Tool_Contributed.SimpleInput()
dbox.set_title("Deep Thought Emulator")
dbox.set_label("What is the answer to the ultimate question?")
dbox.set_default_value("42")
answer = dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details The label is typically a question or other indication of the user response desired.

set_default_value()

Abstract Sets a default value for the entry field of the SimpleInput dialog box.

Class SimpleInput Class

Syntax simpleinput.set_default_value(response)

Argument	Type	Description
response	string	Default value for user response. If "" or None is passed, the entry field is left blank.

Return Type No return value.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

dbox = Ace_Tool_Contributed.SimpleInput()
dbox.set_title("Deep Thought Emulator")
dbox.set_label("What is the answer to the ultimate question?")
dbox.set_default_value("42")
answer = dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details None.

show()

Abstract Displays the SimpleInput dialog box and returns the user response.

Class SimpleInput Class

Syntax simpleinput.show()

Argument	Type	Description
		no arguments

Return Type string — Text in the entry field (either the default value or a value entered by the user); “None” if the entry field is empty.

Example

```
import Ace_Tool
import Ace_Tool_Contributed

dbox = Ace_Tool_Contributed.SimpleInput()
dbox.set_title("Deep Thought Emulator")
dbox.set_label("What is the answer to the ultimate question?")
dbox.set_default_value("42")
answer = dbox.show()
```

Sample output for this example appears in Figure 17-12.

Details This call is blocking.