Packet Tracer 5.0

Inter Process Communication (IPC)

Specifications Document

Modification History

Revision	Date	Originator	Comments
1	January 10, 2008	Michael Wang (miwang@cisco.com)	Creation for general public
2	April 8, 2008	Michael Wang (miwang@cisco.com)	Updated all sections from latest PT version
3	July 3, 2008	Michael Wang (miwang@cisco.com)	Updated all sections from latest PT version

Table of Contents

1. Introduction

1.1. Goal

This document explains the specifications of Inter Process Communication (IPC) features that are added to Packet Tracer 5.0.

1.2. Audience

The scope of this document is intended for Packet Tracer 5.0 External Application developers. It is used to validate requirements and describes detail implementation issues.

1.3. Abstract

This document describes the IPC features of the Packet Tracer 5.0 and details about the architectural design.

The IPC feature targets an external programming extension for Packet Tracer 5.0. The features of PT like creating devices, linking devices etc. and the simulation of sending packets are to be done externally by invoking appropriate API functions. The results of invoked functions are to be sent back to the invoking client. The new feature can support synchronous and asynchronous invocation patterns from the client side.

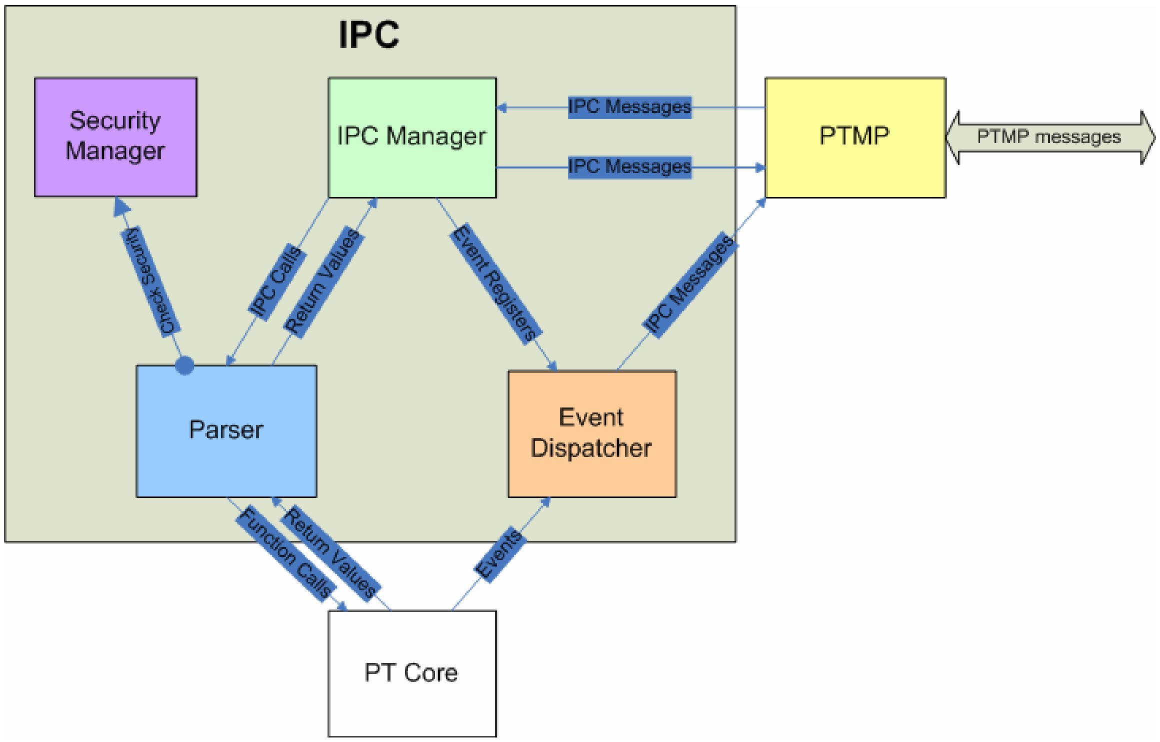
The external API can support multi threading on the client side.

The feature supports authenticated and encrypted channel for secure communication.

The external application communicates to the PT instance using PTMP. Multiple PT instances and multiple external applications running in the same computer are supported under this feature.

The IPC feature enables a user/programmer to develop external applications without depending on the API or library provided. Instead the protocol specification is published. The programmer can connect to the PT instance listening socket for IPC and communicate over this published protocol and make use of the full features of IPC. Any implementation language that supports sockets can communicate with PT's IPC.

2. Architecture



2.1. Communication channel between PT and External Applications

Communication between PT and external applications uses PTMP. PTMP has the concept of server and client. In the case of IPC, PT is the server and the external application is the client. PT is set to listen on a TCP port by default. This will allow external applications to connect to PT without any user interactions. All communication between PT and external applications after the initial TCP connect rely on PTMP.

2.2. IPC Manager

The IPC Manager is the main component in IPC. It controls the incoming external application connections as well as serializing the calls to the Parser component. The IPC Manager has a PTMP Server object that listens to a specific TCP port. Any new connections coming in would create new PTMP Connection objects. After the connections have been authenticated and accepted, they are stored in the IPC Manager in their corresponding external applications that are authenticated into. The external application objects in the IPC Manager have already been registered in PT by the user. They contain the information to authenticate the connections, as well as security and other settings.

PTMP messages coming into these PTMP connections would be forwarded to the IPC Manager. The IPC Manager and calls to IPC from multiple external applications are not done in parallel and uses only one thread. Thus, it keeps a queue of these messages to be processed in sequence. IPC messages of the type IPC Call would be sent to the Parser. IPC messages of the type Event Register would be sent to the Event Dispatcher.

2.3. Parser

The Parser receives IPC calls from the IPC Manager, and calls the PT Core directly on those functions. It also returns the values back to the IPC Manager to send back to the external applications. The work of the Parser is actually integrated and distributed throughout the PT Core. This architecture allows for object oriented structure as well as distributed maintenance.

2.4. Event Dispatcher

The Event Dispatcher is responsible for registering external applications for specific events and dispatching events to the registered external applications. Events from the PT Core are directly sent to the Event Dispatcher. These events include different events happening in the GUI and engine. Some examples are file opened, file saved, device created, device deleted, simulation mode entered, device power change, port IP address changed, etc.

2.5. Security Manager

The IPC opens up many functions in PT. Some calls merely return some state of the network of PT. However, some calls can change the current network in PT, save files, and even make multi-user connections. Therefore, it is needed to have different security settings for different external applications so that some have the privilege to access more powerful IPC calls. The Security Manager checks the security requirement of IPC calls against the security settings allowed for the external application. If the IPC call is allowed, it would execute the call. If it is not allowed, an error would be returned back to the external application. The Security Manager integrates with the Parser throughout the PT Core for the same reasons as the Parser.

The security settings allowed for an external application are listed below. They can be selected independently.

Security Privileges	**Description**
Get network info	Get information and configuration on current network
Change network info	Change information and configuration on current network
Simulation Mode	Switching between Realtime and Simulation modes
Misc GUI functions	Examine device tables
File functions	File new, open, save, save as
Change user preferences	Change sound and display preferences
Change GUI	Add and remove buttons to GUI

Activity Wizard	Use Activity Wizard to author or edit activity files
Multi-user	Makes multi-user connections
IPC	Launch external applications, disconnect external applications
Application	Exit application, change UUID

2.6. External Application Authentication

Different external applications can have different security settings and have different privileges in PT. Therefore, we need a secure mechanism to authenticate external applications when they connect to PT. Before an external application can connect to PT, the user must register the external application in PT. All external applications, when distributed, comes with a meta file that contains information about the external application. This information includes the name, ID, key, relative path to the executable of the external application, and security settings. The user would import this meta file into PT and register the external application to be used.

When the registered external application connects to PT, it sends the ID and key to PT. PT looks them up in the registered external application list for authentication. In order to prevent other external application from using the same ID and key to gain more security privileges, the key must be secure and known only between PT and the external application. To archive this, we provide a utility to the external application developer to encrypt the meta file. The encryption and decryption method is only known to PT and the utility so no one else other than PT and the external application would have access to this key. Because the ID is used as an identifier for all possible external applications that can be registered to PT, it needs to be a unique identifier. We recommend the external application developers to use the hierarchical naming pattern, e.g. net.netacad.cisco.autoIP.

The meta file before passing it to the encryption utility should have the following format in XML:

```
<PT_APP_META>
	<PT_VERSION>5.0</PT_VERSION>
	<IPC_VERSION>1.0</IPC_VERSION>
	<NAME>Auto IP Assign</NAME>
	<VERSION>1.0</VERSION>
	<ID>net.netacad.cisco.autoIP</ID>
	<DESCRIPTION>This CEP assigns IP addresses to devices automatically.</DESCRIPTION>
	<AUTHOR>Michael Wang</AUTHOR>
	<CONTACT>miwang@cisco.com</CONTACT>
	<EXECUTABLE_PATH>autoip.exe</EXECUTABLE_PATH>
	<KEY>cisco</KEY>
	<SECURITY_SETTINGS>
		<PRIVILEGE>GET_NETWORK_INFO</PRIVILEGE>
		<PRIVILEGE>CHANGE_NETWORK_INFO</PRIVILEGE>
		...
	</SECURITY_SETTINGS>
	<LOADING>ON_STARTUP</LOADING>
	<SAVING>NEVER</SAVING>
	<INSTANCES>1</INSTANCES>
</PT_APP_META>
```

The last field in the meta file indicates how many instances of this external application are allowed to connect to a PT instance. PT will not allow more than this number of instances of this external application to connect to it.

2.7. External Application Security Implications

Authenticated external applications are stored in a PT4.conf. To ensure no malicious activity, this file must be made read-only locally. User permissions are managed using the operating system.

2.8. External Application Loading and Saving

External applications can be launched outside of PT and connected to PT manually. PT also provides the option to launch external applications from within PT. The option for external application loading has following three types:

- ON_STARTUP: when PT launches, it will also launch this external application
- ON_DEMAND: PT launches this external application when a file indicates loading it or another external application launches it
- DISABLED: PT will not accept connections from this external application

External applications connections existing at the time of saving can be saved with PT files. Saving the external application connection includes the ID of the external application, instance number of the external application, and any information the external application would like to save about this connection.

The default of saving an external application connection includes only the ID of the external application. In order for an external application to save other information, it needs to register itself for an onSave event in the IPC Manager class. When PT saves a file, it checks to see if any external application is to be saved. If that external application is registered to that event, it will send that event to the external application asking the external application to send the save information back to PT. The save information is appended to the end of the PT save file and needs to be a valid XML tree.

When loading a file with saved external application connections, PT will launch the external application with the ID, and will pass any saved information to that CEP when it is connected.

2.9. Launch Cases

Some use cases of external application requires PT to run without a GUI. PT can be launched without a GUI with the option --no-gui in the command line. However, in some use cases, the external application cannot launch PT with a command line option. To allow for that, we provide a new file type (.pkz) that PT will be able to opened.

PT will also accept option in the command line for the IPC listening port: --ipc-port <port>. If this port is already occupied, it will keep trying incremental ports.

PT will also support the option to take in an UUID for the new instance of PT: --pt-uuid <uuid>. External applications launching PT with this UUID can use it to find the right PT instance.

The following illustrate the launch scenarios of PT and external application instances.

2.9.1. Launch PT and External Applications Independently

Both PT and external applications are launched independently. The user can find out which port PT is listening on for IPC. The user enters that information in the external applications to manually connect to PT.

2.9.2. PT Launches External Application

When PT launches an external application, PT will provide the IPC listening port to the external application as a command line argument. Therefore, all external applications need to accept a command line argument of --pt-ipc-port <port>. The launched external application will use this port to connect to the PT instance that launched the external application.

2.9.3. External Application Launches PT with Command Line

When an external application with access to command line launches PT, it can specify PT IPC listening port and UUID options. Specifying these options, the external application can start trying to connect to a PT instance using the specified port number. For each port number that makes a successful PTMP connection, it will try to check to see if the PT instance has the same UUID as the specified one. If not, it would keep trying incremental port numbers.

2.9.4. External Application Launches PT without Command Line

An external application without access to command line can be a web browser, Flash application, or any other application that launches PT using the associated file type in the operation system. In this case, the external application cannot provide the port number and UUID to the new PT instance. The external application can still connect to the newly launched PT instance or some other available PT instance. It will start trying from the default port. For each port that makes a successful PTMP connection, it will query PT about its opened file and other information about the PT instance. If it is matches the desired criteria, the external application will stay connected to this PT instance. Otherwise, it will keep trying incremental port numbers.

3. Modeling

This section goes into details about the IPC part of PTMP. The types in the messages are using PTMP, with a few extensions listed below:

Name	**Type Value**	**Encoding**
void	0	N/A
byte	1	Same as PTMP
bool	2	Same as PTMP
short	3	Same as PTMP
int	4	Same as PTMP
long	5	Same as PTMP
float	6	Same as PTMP
double	7	Same as PTMP
string	8	Same as PTMP
QString	9	Same as PTMP
IP address	10	Same as PTMP
IPv6 address	11	Same as PTMP
MAC address	12	Same as PTMP
uuid	13	Same as PTMP
pair	14	<first type><first value> <second type><second value>
vector	15	<type><vector size><element1><element2>...
data	16	<custom data type in string>\0 <value>

3.1. Authentication

The IPC Manager has a look up function for the PTMP callback functor to find the password (key) based on the user name (ID of external application).

3.2. IPC Call

IPC call message format:

- Call ID (int): an ID to differentiate between IPC calls
- Multiple calls of the following format:
    - Call name (string): name of the call
    - Multiple arguments of the following format:
        - Type (byte): type of the argument
        - Value (variable based on type): value of the argument
    - End call value (byte): void type value

IPC call return message format:

- Call ID (int): same ID as the call
- Return type (byte): type of the return value
- Return value (variable based on type): return value of the IPC call

The return values to an IPC call can be basic types or data types (16) of the above table. It is depended on the IPC call definition.

Although the IPC calls do not look intuitive, the external application side should have a layer of PTMP and IPC utilities to make the calls easier. The following shows the uses on the external application side and what that layer does in the background.

External Application Code	PTMP and IPC Layer Functionality
Device* device = getDevice(“Router0”);	Stores the IPC call in the device but do not send out the call. “Network().getDevice(Router0)”
Port* port = device->getPort(“Ethernet0”);	Stores the IPC call in the port for both getting the device and getting the port, but do not send it out. “Network().getDevice(Router0).getPort(Ethernet0)”
IpAddress ip = port->getIpAddress();	Sends out the IPC call “Network().getDevice(Router0).getPort(Ethernet0).getIpAddress()”, blocks the thread, waits until PT returns the value, and returns the value to the call.
port->setIpSubnetMask(“1.1.1.1”, “255.0.0.0”);	Sends out the IPC call “Network().getDevice(Router0).getPort(Ethernet0).setIpSubnetMask(1.1.1.1, 255.0.0.0)”, blocks the thread, and waits until PT returns.

3.3. Error Handling

IPC call error return message format:

- Call ID (int): same ID as the call
- Error in class (string): name of the class which the error occurred
- Error name (string): error message

3.4. Events

Event message format:

- Event ID (int): an ID to differentiate between events
- Class name (string): the class that generated the event
- Object UUID (uuid): the UUID of the object
- Event name (string): the event name
- Multiple info of the following format:
    - Type (byte): type of the info
    - Value (variable based on type): value of the info
- End event value (byte): void type value

Event subscription message format:

- Class name (string): the class that generated the event
- Object UUID (uuid): the UUID of the object; if the UUID is null, then it will subscribe to all instances of the same class
- Event name (string): the event name
- Subscribe (bool): true = subscribe, false = unsubscribe

3.5. IPC Calls

The API (all definitions of IPC calls) can be found at the .pki files of specific classes.