# Introduction

iTAT (Integrated Telegraf Agent for TouchDesigner) is runs a Telegraf agent as a subprocess of TouchDesigner. Upon receiving the first batch of metrics, it will automatically build out an HTTP API based on the host system. The API is accessible at `http://<host-IP> :10180/api/<metric url>`. The metrics are also accessible as a list of dictionaries (based on Telegraf’s JSON output), or a prettified version as a dictionary keyed by each metric name.

# Installation

To install the telegraf agent, simply copy the InfluxData folder found in /iTAT/telegraf into your Program Files directory.

# Starting Telegraf

To use the telegraf\_agent .tox, it is preferable to drag and drop the file into the `External .tox` parameter of a baseCOMP instead of dragging the file directly into the TouchDesigner network. It is also (probably) necessary to run TouchDesigner as an administrator because it runs .cmd files which may require admin level privileges.

The metric toggle parameters found on the component do not currently have any effect, but will be used in the future to allow the user to customize what metrics are gathered.

# TouchTime Protocol

TouchTime Protocol (TTP) is based on NTP and implements the same calculation.

Clients send messages on port 32401.

Server sends messages on port 32411.

Port numberings cannot be changed.

## Overview

The client sends a Sync Request packet to a server at a known IP address. The Sync Request includes the alias of the client, a timestamp taken when it was sent, and two more timestamp fields to be filled in by the server. Upon receipt, the server takes a timestamp, stores the two known timestamps, takes a third timestamp, and sends it back to the client.

## Packet Definitions

### Sync Request

The client sends a SyncRequest packet to the server at a known IP address.

JSON Payload:

{

“type” : “01”,

“client” :<alias of the client>,

“timestamp0” : <timestamp>,

“timestamp1” : 0,

“timestamp2” : 0

}

### Sync Response

The Server sends a SyncResponse with two additional timestamps—the time the SyncRequest arrived, and the moment before it sends the SyncResponse. Under the hood, the JSON that’s unpacked from the SyncRequest is updated with timestamps1 and 2 and then repacked.

JSON Payload:

{

“type” : “02”,

“client” :<alias of the client>,

“timestamp0” : <timestamp>,

“timestamp1” : 0,

“timestamp2” : 0

}

### Delay Response

After the client has calculated time offset and network latency, it updates the server with these values by sending a Delay Response packet.

JSON Payload:

{

“type” : “03”,

“client” :<alias of the client>,

“timestamp” : <timestamp>,

“delay” : <latency as calculated by clint>

}

### AbsTime

Server sends its absTime.seconds member to the client.

JSON Payload:

{

“type” : “03”,

“client” :<alias of the client>,

“timestamp” : <timestamp>,

“delay” : <latency as calculated by clint>

}

Touch Monitoring Protocol by default uses

22199 for the server. 22099 for clients. Clients can change in case of multiple instances on the same host

Touch Discovery Protocol

Configurations should be available for p2p and client/server

Client-Server Architecture for InterTouch or TouchIntegrator (TOUCHNET)

# TouchNet Server

Once TouchDesigner instances have found each other and, optionally, synchronized, the main TouchNet server module takes over. The first step is to collect project data and exposed parameters. From there on, the TouchNet server is capable of monitoring and controlling client TouchDesigner sessions.

## Packet Definitions

### Initial Data Request

{ “type” : 11 }

### Initial Data Response

{

“type” : 12,

“alias” : <client alias>,

“projectInfo” : {

<dictionary containing project info>

}

# TouchNet Client

## Packet Definitions

### Project Update

{ “type”: “update”,

“alias”: <client alias>,

“params”: {

<type of info>:<value

}

}

# UI Connection

The control UI shall be completely separate from the master TouchDesigner node/TouchNet server, and as such must communicate with the server over a network connection. If a connection is present, the server shall update the UI of any changes in applicable data without being asked. The UI will also announce its presence on the network by joining the TouchNet Discovery multicast group 239.255.22.22. However, it will send data its discovery announcement to 2879.

Primarily Unicast

Sends on TouchNet Port, by default 22099, can be changed if multiple instances on same host

1. Server sends packet requesting initial default project data.

Packet Structure in JSON: {

Type: 11,

}

1. Client sends initial data, packet type 12.
2. Server sends request for ExposedParameters type 13
3. Client sends ExposedParameters, packet type 14

Server sending action request:

Packet Type= 15 or 17

{type:15,action:play, etc, params:[]}

{type:17,member:perform, etc, val:}

# Health Monitoring

All clients will send a ping to the server, including the UI, every 10-15 seconds. The server will periodically check how long it’s been since a heartbeat was last sent.

## PacketDefinitions

### Heartbeat

{

“type” : ”ping”,

“alias” : <alias of the client; “UI” will use alias “UI”>

}

# Data Storage

The TouchNet server will make use of the component storage dictionary as a database for the clients, their network settings, project settings, exposed parameters, and health, as well as the status of the UI.

## Dictionary Structure

{ “Clients” : {

<Client Alias> : {

{ “NetworkInfo” : {

“ipaddress” : <ip>,

“port” : <port>

}

},

{“ProjectInfo : {} },

{“ExposedParameters” : {

<unique name> : {

“path” : <absolute path to operator>,

<unique name of parameter> : <par value>

}

}

}

}

}

# UI

TouchNet is controlled from a browser-based UI, currently displaying a list of nodes available for control and basic actions (Perform, Play, Stop, Save, Quit). The UI uses Vue.js to be responsive and update its list of available nodes, perform state, and node status data.

The Vue instance is initialized with a list of nodes, each element being a JavaScript object of the following form:

{ name:<alias>,

ip:<ip address>,

rate:<frame rate>,

perform:<perform mode (boolean)>

}

12. Components

12.1 TouchNetDiscovery

13. JSON-RPC API

Action requests:

{“jsonrpc”: “2.0”,

“method”: <method>,

“id”:<id>

11.1 Web API

11.1.1 User Authentication

Method: GET

URL: http://<server address>/login

DATA = {username=username,password=<hashed password>}

The server will return a session cookie in the form of a token in the response header

11.1.2 Controls

Method: GET

URL: http://<server address>/control?username=username&token=token

Parameters: username, token

11.1.3 Update Nodes

Method: GET

URL: /nodes? username=username&token=token

This component can be used in either Server Mode or Client Mode. It is used for finding peers on the network and initializing connection between servers and clients.

NOTES:

1. If no Alias name, reject packet
2. Changing to Client mode no longer enables the COMP. I think a change from one mode to the other should require the user to explicitly Enable the device, since its proper behavior may require other parameters to be set first, namely, the Alias while in Client mode.
   1. When switching to Serve mode, let’s still store the Client alias. This is relatively simple.