Tangent API

A simple and efficient way to do client server programming.

Tangent API (Application Programming Interface)

COM DLL (32-bit): **TangentAPI.dll**
COM DLL (64-bit): **TangentAPIx64.dll**
API Version: **2010.8.24**
Server ProgID: **TangentAPI.Server**
Client ProgID: **TangentAPI.Client**
Threading Model: **Both**
Supported Languages: **C#, VB.NET, Managed C++, Visual C++, VB6, Jscript, VBScript, and any other language that supports COM.**

This document describes Tangent API mostly from the perspective of .NET.

Server Methods:

void **Startup** (string **ipAddress**, short **port**, int **maxClientConnections**);
Starts up a server on the specified IP address and TCP port. Once started up, the server begins listening for new clients to connect. Also, once started up, the server is ready to send and receive data between connected clients. After the maximum number of client connections specified has been obtained, the server stops listening for new clients. If a client subsequently disconnects, therefore freeing up a client connection slot, the server then begins listening for new clients once again.

void **ShutDown** ();
Shuts down a server. The server stops listening for new clients, and all clients that are connected get disconnected.

void **SendDataToClient** (int **clientID**, object **data**);
Sends data to a specified client that is connected to the server. The data is of type object, which allows a number of types as described in the Object Data section.

void **SendDataToAllClients** (object **data**);
Sends data to all clients connected to the server. The data is of type object, which allows a number of types as described in the Object Data section.

object **GetConnectedClientIDs** ();
Retrieves client IDs of all the clients that are connected to the server. The object returned is of type object[] array, and each individual element in this object[] array is of type int that specifies a client ID. Returns a zero-length object array if no clients are connected. In C++, the return type is a VARIANT which is of type VT_ARRAY | VT_VARIANT (a SafeArray of variants), where each array element is a VARIANT of type VT_I4.

void **DisconnectClient** (int **clientID**);
Disconnects the specified client from the server.

void **StartBroadcastingIPAddress** (short **udpPort**);
Broadcasts the IP address of the running server on the specified UDP port. Once broadcasting starts, clients can get the IP address of the server. After a client obtains this IP address, the client can then use it to facilitate a connection to the server. This allows a "zero-configuration" client to connect to a server, meaning, the IP address does not have to be known or configured for a client in order for it to connect to a broadcasting server.

void **StopBroadcastingIPAddress** ();
Stops broadcasting the IP address of the server.

void **StartListening** ();
Causes the server to start listening for new clients to connect. When Startup() is first called, the server automatically starts listening, so it is not necessary to call this method after the first startup. This method is useful after StopListening() is called.

void **StopListening** ();
Causes the server to stop listening for new clients to connect. The server will not listen for new clients, even if Startup() is subsequently called. StartListening() can be called to allow the server to start listening for new clients again.

object **GetValidIPAddresses** ();
Retrieves valid IP addresses found on the system. The object returned is of type object[] array, and each individual element in this object[] array is of type string that specifies an IP address. Returns a zero-length object array if no valid IP addresses are found on the system. In C++, the return type is a VARIANT which is of type VT_ARRAY | VT_VARIANT (a SafeArray of variants), where each array element is a VARIANT of type VT_BSTR.

Server Properties:

bool **IsRunning**
Returns whether the server is currently running or not. A server that is running is ready to send and receive data between any connected clients.

bool **IsListening**
Returns whether the server is currently listening for new clients to connect or not.

int **SendReceiveTimeout**
Gets or sets the send/receive timeout in milliseconds. The send/receive timeout applies to the timeout for sending and receiving data. The default value is 5000, or 5 seconds.

int **ConnectionTimeout**
Gets or sets the connection timeout in milliseconds. The connection timeout is used when a client attempts to connect to the server. The default value is 5000, or 5 seconds.

Server Events:

void **ConnectedClient**(int **clientID**);
Occurs when a client has connected to the server.

void **DisconnectedClient**(int **clientID**);
Occurs when a client has disconnected from the server.

void **ReceivedDataFromClient**(int **clientID**, object **data**);
Occurs when the server has received data from a client. The data is of type object, which allows a number of types as described in the Object Data section.

Client Methods:

void **ConnectToServer** (string **ipAddress**, short **port**);
Connects the client to a server that is running on the specified IP address and TCP port.

void **DisconnectFromServer** ();
Disconnects the client from the server.

void **SendDataToServer** (object **data**);
Sends data to the server. The data is of type object, which allows a number of types as described in the Object Data section.

string **GetServerIPAddress** (int **udpPort**);
Retrieves the IP address from a server that is broadcasting its IP address. A server must be broadcasting its IP address on the specified UDP port in order for this method to succeed.

Client Properties:

bool **IsConnected**
Returns whether the client is connected to a server or not.

int **ClientID**
Returns the client ID assigned by the server. The client must be connected to a server when this property is accessed, otherwise an exception is thrown.

int **SendReceiveTimeout**
Gets or sets the send/receive timeout in milliseconds. The send/receive timeout applies to the timeout for sending and receiving data. The default value is 5000, or 5 seconds.

int **ConnectionTimeout**
Gets or sets the connection timeout in milliseconds. The connection timeout is used when the client attempts to connect to a server. The default value is 5000, or 5 seconds.

Client Events:

void **ReceivedDataFromServer**(object **data**);
Occurs when the client has received data from the server. The data is of type object, which allows a number of types as described in the Object Data section.

void **DisconnectedFromServer**();
Occurs when the client has disconnected from the server.

Object Data:

Data that is sent and received is of type object, which allows the following types to be sent and received: bool, byte, char, date, double, int, uint, short, ushort, long, ulong, single, string, byte[] (byte array), object[] (object array, where the objects can be any of the types specified here including arrays). In C++, the following types can be sent and received: VARIANT of type VT_UI1, VT_I1, VT_UI2, VT_I2, VT_UI4, VT_I4, VT_R4, VT_R8, VT_DATE, VT_BSTR, VT_ERROR, VT_BOOL, VT_I8, VT_UI8, VT_INT, VT_UINT, VT_ARRAY | VT_UI1 (SafeArray of bytes), VT_ARRAY | VT_VARIANT (SafeArray of variants) where each element is a variant that can be of any of the types specified here. VT_DISPATCH is not allowed. There is a maximum length allowed when sending data, and an error will result if this limit is exceeded. For a string (or VARIANT of type VT_BSTR), the maximum length is 0x007FFFFC characters. For a byte array (or VT_ARRAY | VT_UI1), the maximum bytes that can be sent is 0x00FFFFF9. For an object array (or VT_ARRAY | VT_VARIANT), the maximum objects that can be sent is variable, depending on what types of objects are in the array.

Tangent API Threading Model:

Tangent API is a COM DLL that registers with the threading model “Both”. This means that it can be used in either a single-threaded apartment where event callbacks are synchronized on the GUI thread, or a multi-threaded apartment where callbacks are not synchronized thus providing better performance.

Single threaded apartment or STA, allows event callbacks to be synchronized automatically for you via Windows messaging. In .NET, this allows a developer to modify Windows Form members from within Tangent API’s event callbacks without having to invoke upon the GUI thread. By default,.NET projects are configured using the STA model. This is configured by default in the Program.cs file with the [STAThread] attribute (in C#). The STA model makes Windows Forms development easier, because the developer is ensured that event callbacks run on the same thread as the GUI, and thus GUI members can be freely modified from within callbacks. With STA, the developer is freed from the complexity of thread synchronization.

While STA makes development easier in Windows Forms programming, it is not as efficient as the multi-threaded apartment model or MTA. With MTA, methods can be called from any thread freely and event callbacks are fired instantly from whichever thread they are on. This does mean the developer may need to provide any data synchronization. However, MTA provides the ability to run a server that provides the most efficiency and high-performance, because method and event callbacks do not need to be synchronized using Windows messaging as with STA. In C# .NET projects, using the MTA model is as simple as changing the [STAThread] attribute to [MTAThread] in Program.cs. In C++, the threading model is determined by the second parameter of the function CoInitializeEx, using either COINIT_APARTMENTTHREADED for STA or COINIT_MULTITHREADED for MTA.