Getting Started with the Thermal Desktop Application Programming Interface (TD API)

# Introduction and Prerequisites

The Thermal Desktop Application Programming Interface (TD API) allows you to automate many of the tasks currently performed interactively using TD’s Graphical User Interface (GUI). The API gives you the tools to programmatically create, query, edit, delete, and run models. You can use any .NET language to interact with the API (C#, VB.NET, F#, etc.) or any system that can load .NET assemblies, e.g., Matlab.

Regardless of how you interact with the API, you’ll need to have at least an intermediate understanding of .NET object-oriented programming. If you are starting from scratch, we recommend learning C#, since most of our examples are written in that language. There are many excellent books and web tutorials available for learning C#[[1]](#footnote-1). For the remainder of this document, we will assume you have an intermediate knowledge of C# and familiarity with the .NET global assembly cache (GAC).

The API is installed beginning with TD 6.0, which was released in July 2017. You’ll need to upgrade to at least TD 6.0 to use the API.

The API targets .NET version 4.0, so if you’re using Visual Studio (VS) you’ll need VS 2010 at least.

# Hello World

Let’s create a simple C# program using the TD API. Start by creating a C# Console Application in Visual Studio. Next, add a reference to the latest TdApi assembly (dll), which you can find in the GAC. (Try looking in C:\Windows\Microsoft.NET\assembly\GAC\_MSIL.) The assembly will have a name like TdApiV?.dll, where “?” is the version number. You might find multiple versions; use the latest. API versioning is discussed in detail in “A Note on TD API Versioning”, below.

Add the following code to your Main method: (Replace “TdApiV1” with whatever version of the API you’re using.)

static void Main(string[] args)

{

TdApiV1.ThermalDesktop td = new TdApiV1.ThermalDesktop();

td.Connect();

td.Print("Hello World\n");

}

Now try running your program. It should open TD, then write “Hello World” in the AutoCAD console:



Let’s take a look at how it worked. First, we created a ThermalDesktop object called td. This object represents one instance of TD. It has dozens of methods for interacting with TD models. A single TD API client program can create an arbitrary number of ThermalDesktop instances, allowing you to manipulate several models and communicate between them.

Next, we called the td.Connect() method. By default, this will start a new instance of TD using the latest version of AutoCAD installed. You can control how it works, including selecting a dwg file to open, trying to connect to an running instance of TD, and many other options, using the ThermalDesktop.ConnectConfig property.

Connect(), as well as most ThermalDesktop commands, is called synchronously, so it will only return control to your program once it finishes connecting. If there is a problem, it should throw an exception. This is true of all TD API methods.

Once Connect() returned, we called Print and passed it a string, terminated by newline (“\n”). As you probably guessed, this printed the string to the AutoCAD console.

# Creating Nodes and Conductors

Let’s try doing a little more with the API. Here is the simplest possible code to create two nodes joined by a conductor:

using TdApiV1; // replace with whatever version you’re using

// ...

static void Main(string[] args)

{

// start a new TD instance

ThermalDesktop td = new ThermalDesktop();

td.Connect();

RcNodeData node1 = td.CreateNode();

RcNodeData node2 = td.CreateNode();

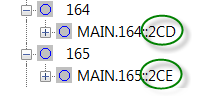
RcConductorData cond = td.CreateConductor(node1.Handle, node2.Handle);

}

We’ve added a “using TdApiV1” statement to avoid prefacing our type names with the TdApiV1 namespace. From now on we’ll just assume that this is present in any example programs.

How did this program work? After starting a new instance of TD, we created two RcNodeData objects, node1 and node2, using the CreateNode() method. You’ll find ThermalDesktop.Create methods for many entity types; for each, it creates the default entity in TD and returns its properties to your client program.

When TD created each node, it set the RcNodeData.Handle property to a unique identifier, the same string you may have noticed in the TD GUI’s Model Browser:



Since TD allows duplicate SINDA names for some entities, the API uses handles to identify most entities uniquely. You’ll notice that the CreateConductor method accepts the node handles to identify which nodes it is attached to.

The previous program created two default nodes at position (0,0,0) and connected them with a default conductor. That’s not too useful! Let’s expand it to edit the node and conductor properties and introduce some new concepts:

static void Main(string[] args)

{

// start a new TD instance

ThermalDesktop td = new ThermalDesktop();

td.Connect();

// set API WorkingUnits and TD units to English

td.SetUnits(Units.Eng);

// create two nodes

RcNodeData node1 = td.CreateNode();

RcNodeData node2 = td.CreateNode();

// update the nodes

node1.Submodel = "helloworld";

node2.Submodel = "HELLOworld"; // case-insensitive

node1.Id = 1;

node2.Id = 2;

node1.NodeType = RcNodeData.NodeTypes.BOUNDARY;

node2.NodeType = RcNodeData.NodeTypes.DIFFUSION;

node2.MassVol = 10.0; // BTU/F

node2.Origin = new Point3d(1, 0, 0); // ft

td.SetRcNode(node1); // this actually updates the node in TD

td.SetRcNode(node2);

// create a conductor

RcConductorData cond

= td.CreateConductor

(node1.Handle, node2.Handle); // use handles to identify nodes

cond.Submodel = "helloworld";

cond.ValueExp.Value = "100.0 + 100.0"; // expressions are strings

cond.ValueExp.units

= new UnitsData(Units.SI); // expressions have own units (G = 200 W/K)

td.SetRcConductor(cond); // this actually updates the conductor in TD

// update the view

td.ZoomExtents();

}

After starting a new TD instance, we called SetUnits to set our units to English. This method does two things: it sets the unit preferences in TD, and it sets a TD API property called Units.WorkingUnits. Every dimensional value you set or get in the API is presented in the current WorkingUnits. For example, if you set WorkingUnits.temp to C, then set the temperature of a node to 0, its temperature will be 0, of course. If you now change WorkingUnits.temp to F and get the temperature of the node, you will find that it equals 32. As mentioned previously, SetUnits changes both the TD unit preferences and the API WorkingUnits. You can change the TD unit preferences independently using SetDwgUnits, and you can change Units.WorkingUnits independently by changing it directly.

Next we created two nodes using CreateNode and updated their submodel, ID, node type, m\*Cp (using the MassVol member), and the origin of node2. Note that MassVol was entered in BTU/F, since we set WorkingUnits to English previously. (This member is called MassVol and not something like “mCp” because it can be used to set node mass, volume or m\*Cp depending on how other node properties are set. It defaults to m\*Cp.) Then we used SetRcNode to update the nodes in TD.

It is vitally important to understand that the RcNodeData objects in your client program are not automatically linked to the nodes in TD. For example, when you change the Submodel property of RcNodeData node1, the change is not automatically propagated to TD. You update the actual node in TD using the SetRcNode method, as shown above. There is a family of Set methods for editing TD entities. (The Set methods can also create new entities with custom properties, rather than calling Create followed by Set. If the handle passed to the Set command is an empty string, then TD will create a new entity and update the handle in the passed Data object, e.g. RcNodeData.) All Data types in the API act this way, i.e., they do not affect the corresponding entities in TD unless you call a Set (or in some cases an Update) method.

Next we created a conductor using CreateConductor and passing it the two node handles, as before. Then we updated the submodel and the value of the conductor, using the ValueExp member. Members that end in “Exp” are strings that set expressions in TD. There is a corresponding RcConductorData.Value member that sets a double, but as in the GUI, setting ValueExp supersedes setting Value. (These members are called Value and ValueExp rather than G and GExp because they can store other physical values for conductors depending on how other conductor properties are set.) Similar to the nodes, once we modified our RcConductorData object in the client program we passed it to SetRcConductor to update the conductor in TD.

At the end of this program we zoomed out to include the whole model in the viewport using ThermalDesktop.ZoomExtents().

# A Note on TD API Versioning

Version 1 (V1) of the TD API was included with TD 6.0 as the TdApiV1.dll assembly installed in the Global Assembly Cache (GAC). All V1 classes are contained within the TdApiV1 namespace, and we promise never to change this interface. We will also include TdApiV1.dll with all future releases of TD. Thus you can write software referencing V1 knowing that it won’t be broken by new releases of the API.

Each subsequent release of the API will be contained in its own dll (TdApiV2.dll, etc.) or family of related dll’s (TdApiV2.Results, TdApiV2.Forms, etc.) and use its own namespace (TdApiV2, etc.). When we develop each subsequent release, we will start with a complete copy of the previous release (with updated namespace), then make changes only as required to implement new features. So most commands should remain unchanged from release to release and you should be able to reference new versions of the API with a minimum of changes to your old programs.

# Exploring the TD API Demos

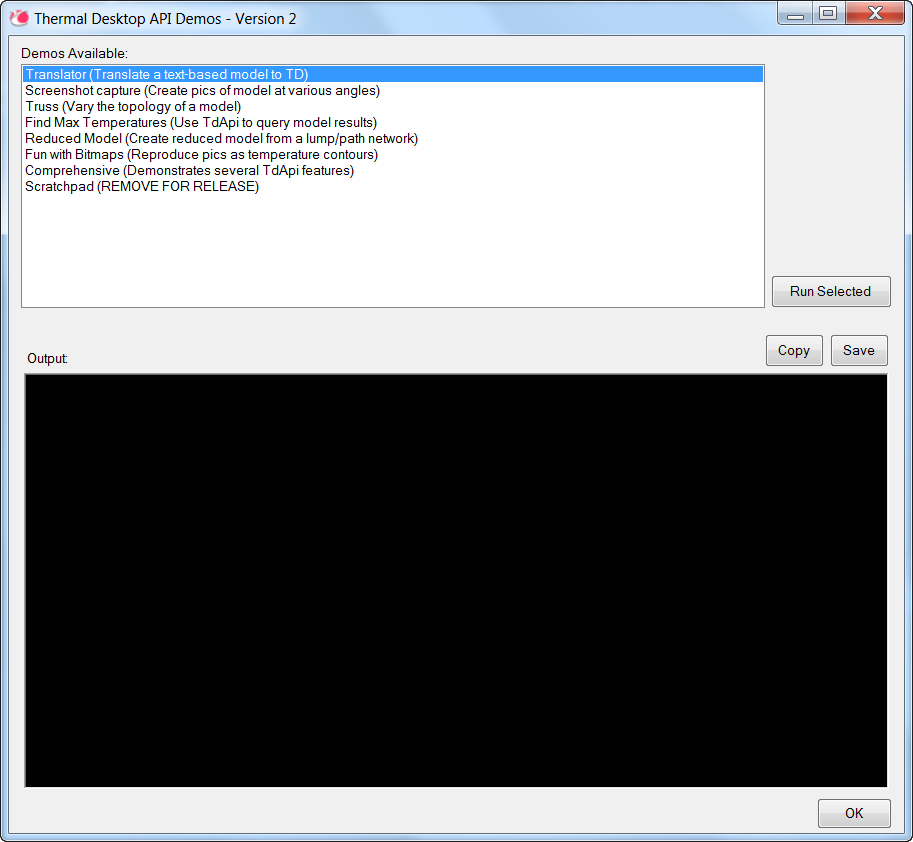
We have included several more advanced TD API demos with your TD installation. The source code can be found in the TD installation directory, usually in the following location:

C:\Program Files\Cullimore and Ring\Thermal Desktop\TdApiDemos

In that directory you’ll find a Visual Studio project file called TdApiV?Demos.csproj, where “?” will be the current version number. Open this in Visual Studio (VS) to begin exploring the demos.

Once you open TdApiV?Demos.csproj in Visual Studio, you may have to update its references to the TdApi dlls. Take a look in the Solution Explorer for the project’s references. For each TdApi reference listed in the References node, if it’s not valid (VS displays a warning icon on the reference), delete it and re-add it from the GAC. Make sure to reference the correct version (V1, V2, etc. in the name).

Once you’ve confirmed that the references are correct, you should be able to build and run the demos, presenting you with a menu that looks similar to this one:



These demos were written using nothing more than the .NET library and TD API. They demonstrate various use cases for the API. Once you’ve tried running them, you can examine the source code using the Visual Studio Solution Explorer.

# Troubleshooting

## SINDA/FLUINT throws an error when I try to run a case using the API

You’ve tried to run a case using ThermalDesktop.RunCase but it doesn’t work, even though you can normally run cases using the GUI. Are you running your client program from within Visual Studio using the debugger or the “Start Without Debugging” command? Try building your client and launching it by double-clicking on the exe file in Windows Explorer instead. Sometimes Visual Studio introduces environment variables into child processes that interfere with SINDA/FLUINT and cause it to fail.

1. One of our favorite C# books is [“Pro C# 2010 and the .NET 4 Platform” by Andrew Troelsen](http://a.co/19EJ2n7). Later editions of this book have slightly different titles; all have excellent reviews. [↑](#footnote-ref-1)