

## **Getting Started – Quick Start**

- In the CTI32 Config utility that loaded automatically after install perform the following:
  - Make sure the Intel / Dialogic driver is running
  - Click the “Reconfigure Boards Section” along the top tool bar
  - Follow the wizard to set up the Intel/Dialogic boards / phone line type (or HMP) in your system
  - After you have completed the wizard, press the Save button on the top toolbar
  - At the bottom of the screen, you will see a button “Start Service”. Click on that button
  - After a few seconds, click on the “View CTI32Engine.Log” button on the top toolbar. Scroll to the end. If everything is working properly, you should see “Waiting for call” on each channel in your system.
  - Dial into the system, you should hear the InboundIVR demo application.

## **Getting Started - Introduction**

CTI32.NET was designed to quickly get you up and running with your telephony application. First let me get you comfortable with the various parts of CTI32. Explore to the \program files\cti32 folder as I describe the main components. There are 3 major components to CTI32.NET.

- 1) CTI32.DLL. This is an unmanaged windows DLL written in C++ that interfaces directly to the various Dialogic API's. This DLL has been field proven since 1995 and installed on thousands of Dialogic ports. In order for this DLL to run, the Dialogic drivers must be installed on your machine. In addition, it depends on the following modules:
  - a. NTGDK.DLL, GLFXIF.DLL, GLFXSTD.DLL – GammaFax DLL's
  - b. SNDEMAIL.DLL – outbound SMTP e-mail sender for reports
  - c. Note: the CTI32.INI is optional and is used to define Voice Recognition and SIP registration properties.
  - d. If you did not install Global Call and depending on the version of Dialogic driver, you may also need a file called libgc.dll. You can contact us to get a copy of that file, or re-install the dialogic drivers with the global call box checked.
- 2) CTI32NETLIB.DLL. This is a wrapper for the CTI32.DLL for .NET languages which has also implemented intellisense. Most of the telephony commands that you will use will be called from this class library.
- 3) CTI32SVC.EXE, CTI32ENGINE.DLL. The CTI32 service is in charge of opening all of the Dialogic channels and creating the threading for each channel and getting ready for an incoming or outgoing call. This saves you all of the time in creating a program framework, opening ports, and figuring out the threading. The source code to the service and engine can be requested for a small fee in case

you want to use it as a reference to create your own framework. However, I would recommend using the engine provided. We have not yet found an application that could not be implemented using the CTI32 engine (including implementing a user interface with statistics fed from the engine)

The CTI32 engine relies on the cti32engine.config XML file for definitions of Dialogic boards, ports and various program options. We have created the CTI32Config.exe to make configuring your specific environment easy. All of the common tasks that you need to do can be done from the CTI32Config utility.

**NOTE: Non- .NET USERS:** You will be interfacing to CTI32.DLL directly. Please refer to the cti32.DLL.pdf documentation. You may want to refer to the cti32engine source code for a template on how to build your own engine in C++, Delphi or whatever threaded environment that you are using. Call us if you would like to view that C# source code to help you create your own engine in the language of your choice.

### ***Getting Started – How the engine works***

The engine depends on one or more user class libraries that you have written. In the default section of the config file you will see several definitions for “Default” (DefaultDLL, DefaultMethod, DefaultType,) several entries for “Dispatch” (DispatchDLL, DispatchMethod, DispatchType, DispatchStopMethod) These define what DLL and method the engine should call to implement your application logic.

The DefaultMethod in the DefaultDLL is called when an incoming call occurs. There is also another optional method defined called the DnisMethod. This method can be defined in the DefaultDLL. This method is called passing the DNIS (called number). This method can implement some logic of yours to determine which DLL and method should be executed based on what number was dialed. This is a cool feature that allows you to implement many different applications on the same box sharing the same hardware all running concurrently. You could define in a table which DNIS will trigger which application.

The DispatchMethod and DispatchDLL give you the opportunity to implement your own “dispatch” logic. (normally used to trigger outbound calls) The engine creates a separate thread at the beginning of the program and executes your dispatch logic. You are expected to run this thread in a loop forever until the DispatchStopMethod is called when the service is stopped. You could use this thread to check your database for a new job of outbound calls. Then this thread could call some methods in the engine to kick off an outbound call on a channel. We have used the dispatch thread to implement many interfaces to external systems. Our Visual Voice like product simply implements a Dispatch thread with remoting to a VB 6 COM object that converts a Visual Voice command to CTI32 command. You can literally implement almost anything in the dispatch thread.

When the CTI32 service is started and assuming you have defined your Dialogic boards correctly using the CTI32config utility, the system is ready for either an incoming or outgoing call.

### ***Getting Started – Incoming call***

- 1) The engine detects the incoming call and retrieves the DNIS, ANI, and CallerId name.
- 2) Optional: Your DnisMethod is called in your DLL passing the DNIS. Your method returns the module, method, and class type of the module the engine should call.
- 3) The engine calls your DLL and method that you specified for the incoming call. It passes a structure defining the entire configuration and a structure defining the phone line the call came in on. (cfg and lineData) You will use these structures all the time for needed information.
- 4) Your DLL method executes your script calling the telephony functions in the cti class (in CTI32NetLib.DLL) Use try/catch blocks to catch the event that the caller hung up the phone.
- 5) When you are finished return and the engine will hangup the phone line.

### ***Getting Started – Outgoing call***

- 1) Your dispatch thread will determine when it is time to issue a phone call and will extract the telephone number from a data source.
- 2) The dispatch thread will call a method in the engine to launch an outbound call. You can have the engine pick a channel for you or specify this yourself. You must pass the DLL, method, and classType for the code you want executed for the outbound script. Example:  

```
cfg.eng.LaunchOutboundCall("YourCodeSample.DLL", "OutboundCall", "YourCodeSample.YourCodeSample", phoneNumber);
```
- 3) The engine will automatically notify the thread responsible for that channel and call the method in the DLL specified.
- 4) Your outbound call script will make the call, check the result and make function calls into the cti class (in CTI32NetLib.DLL)

A sample application is showing these concepts is written for you in either C# or VB.NET. You can find it under the “Samples” folder.

### ***Getting Started – Steps to getting the sample running***

- 1) Run the CTI32Config program. An icon has been installed on your desktop.

- 2) The system has been shipped to run a 4-port analog board. If you have other Dialogic hardware, then press the “Reconfigure Boards Section” button on the top toolbar. A wizard will pop up.
- 3) Specify the number of Dialogic boards in your system and press “Yes”
- 4) Pick the type of board in your system for each board and verify how many ports are on that board. Press Next.
- 5) For now Ignore the Dialing Rule Name.
- 6) If you are using T1 (CAS) you may want to enter the mask for incoming digits.
- 7) Make sure your Dialogic board is installed and the Dialogic driver is running.
- 8) Start the CTI32 windows service in the CTI32Config program. (at the bottom of the screen)
- 9) You can check the logs (“View CTI32Engine.Log” button or “View CTI32.Log” button) or run the Tmonitor program (“Launch TMonitor” button) all from the CTI32Config application.
- 10) Make a phone call. If everything is working correctly, you should hear the demo program and see activity on the Tmonitor. (Launch TMonitor)

### ***Running the samples***

You can view and run the various samples by pressing the “Samples” button on the top toolbar.

Select the sample the most closely matches your application.

Press the Save button

Stop and start the CTI32 service so that it will re-read the new CTI32Engine.config file.

### ***Modifying the samples to suit your own application***

Next, let's look how you can modify a sample application for your own use. There is source code for all the samples under the Samples folder in the installation folder. Most of the samples will have both C# code and VB.NET code.

- Open Visual Studio.NET 2003. File / Open Solution. Choose Sample.sln.
- Each sample is a class library DLL. Click on the Solution Explorer and you will see there is a C# or vb code file. It also has two references to CTI32Engine and

the CTI32NetLib. NOTE: When modifying any of the source code, you may have to delete and re-add the References to CTI32Engine and CTI32NetLib. This is because Visual Studio.NET saves the full paths to the References and your installation folder is not the same folder that I used to Add References.

- Look at the code and look how easy it is to write a CTI32 application.
- You will notice vast information of help and documentation in the intellisense. Find a line and type cti. (cti dot) You will see all of the telephony methods, properties, and enums. Select a method like Play and it will walk you through all of the parameters and what they mean. Cool huh?
- Try compiling to make sure everything compiles OK.
- To run your changes, you will need to stop the CTI32 service. You can do this by pressing the “Stop Service” button at the bottom of the configuration utility screen. (CTI32Config). Then copy in your new DLL and press “Start Service”.

## ***Files Installed***

Now I'd like to explain the files copied into the Installation folder and what it is for:

- CTI32NetLib.DLL – This is the C# class library that has all of the cti methods described in the CTI32.Doc file. Please note that the .NET methods have dropped the CTI prefix on the method names as described in the CTI32.doc file.
- CTI32.DLL – This is the original unmanaged CTI32 library that talks to the Dialogic boards. The CTI32.DLL library has been available since 1995. It is a field proven, high density, reliable telephony platform for Dialogic equipment. There are thousands of ports in use using the library. One of the goals of the .NET version was not to lose that field proven experience. This is done by minimizing the changes – so this is why we kept CTI32 around. This CTI32.DLL is the same version shipping with the C++ version. Optionally, the CTI32.DLL will read CTI32.INI for added parameter control for voice recognition and SIP register information.
- CTI32Engine.DLL – This is the “engine” used by the cti32svc application to start all of the dialogic boards and channels. The cti32svc windows system service simply calls the Start and Stop method in the engine.
- CTI32SVC – a windows service application that calls into the CTI32Engine class. This service also requires cti32svc.exe.config and cti32svc.InstallState.
- Cti32engine.config – this is the XML configuration file read by the CTI32Engine to learn the rules for execution and which boards and ports to open.

- SNDEMAIL.DLL – Our unmanaged DLL used by CTI32.DLL to send e-mails and the status report.
- NTGDK.DLL, GLFXIF.DLL, GLFXSTTD.DLL. These are DLL's provided by Dialogic / GammaLink. They are needed for the Fax Functions of CTI32. They are included because they are not installed by default from the Dialogic drivers. All other required dialogic DLL's are installed by the Dialogic Service Releases.
- ENGLISH.VAP. This is a file that contains a bunch of WAVE files. It is a whole bunch of WAVE files indexed into one file. These contain the necessary voice segments to play numbers, dates, times, etc.
- TMonitor.EXE – we've also talked about this. It is a windows application that can watch the phone lines and debug log messages. This also uses CTI32StatusClient.DLL that communicates to the CTI32 service. You can also use this CTI32StatusClient Class with your applications to get information from the service. The Logging.DLL file is used with the CTI32StatusClient.DLL.
- CTI32Config.EXE – the configuration program that writes out the Cti32Engine.Config XML file. This program also requires AxInterop.SHDocVw.DLL and Interop.SHDocVw.DLL.
- CTIUtility.DLL. This is a DLL called by the CTI32 service and engine when Tmonitor gives it a Dial or "Line Tap" command.

That's the overview. Go ahead and get started with your app. You can call with any questions (303) 689-0720. I will do my best to support you free of charge during the 30 day evaluation period and 90 days after your purchase. In the event, that your question becomes too involved, I may have to charge a fee to cover my time in supporting you. I don't do this very often, but when you have telephone line installation issues, or small change requests – I sometimes have to cover my time.

## ***Distribution Files***

Here is the description of the distribution folder structure and their contents:

### Documentation

Cti32.DLL.doc – the original documentation to the CTI32.DLL. The .NET object browser or intelli-sense will show you any differences to the original documentation. Note: The prefix CTI on each of the methods have been dropped on the .NET version.

Cti32NetLib Documentation.chm – Help file for CTI32.NET and the .NET cti class in CTI32NetLib

Cti32Engine Documentation.chm – Help file for the CTI32Engine object.

Getting Started.doc – this document

Release Notes.doc – changes for this version of CTI32 and a revision history.

Cti32Guide.pdf – describes sample applications and random CTI32 topics

CTI32Config.pdf – Describes how to use the configuration utility

#### Samples

This folder contains sample “user” or application source code.

Examples for VB and C# are included.

#### Source

Full source code (if purchased) for the CTI32 product. All modules use the VS.NET 2003 compiler.