EZTapiAPI ™
Software Reference Manual

Version 2.02

For Windows Based Operating Systems

By PC Phone Connections

Trademarks
EZTapiAPI ™ SDK, and PC Phone Connections Logo are trademarks of PC Phone Connections.

Windows95®, Windows 98®, Windows NT®, Windows 2000®, and Windows XP® are trademarks of Microsoft Corporation.

All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Part Number
Part Number 202
Revision 02

License Section
The granting of any license and the terms of any warranty is explicitly defined in the Software License Agreement for this product.

1. Software License Agreement

PC PHONE CONNECTIONS
SOFTWARE LICENSE AGREEMENT

IMPORTANT — READ CAREFULLY BEFORE USING SOFTWARE

This Software License Agreement ("Agreement") is a legal agreement between you and, if applicable, your company ("you" or "Licensee"), and Robert Bamberg d/b/a PC Phone Connections ("Licensor"). The Licensor is the manufacturer of this software product ("Software"). The Software, as that term is used in this Agreement, includes EZTapiAPI SDK software (including all libraries, dll files, any additional programs, updates, modifications, fixes, patches, upgrades, and revisions supplied initially or subsequently to the Licensee), the associated media, any printed materials, and any "online" or electronic documentation. Any software provided along with the Software that is associated with a separate end-user license agreement is licensed to the Licensee under the terms of that license agreement. By installing, copying, downloading, accessing, or otherwise using this Software, the Licensee agrees to be bound by the terms of this Agreement and the Licensee represents that he or she is authorized to accept the conditions of this Agreement individually and, if the Software is to be used by the Licensee's company, on behalf of the Licensee's company. If you do not agree to the terms of this Agreement or if you do not have the authority to agree to this Agreement, you may not use or copy the Software, and you should destroy the Software and notify Licensor in accordance with Section 7 of this Agreement. This Agreement is valid and grants the end-user rights ONLY for evaluation purposes unless a valid license key is provided by Licensor or an authorized vendor.

BY INSTALLING, COPYING, DOWNLOADING, ACCESSING, OR OTHERWISE USING THE SOFTWARE YOU INDICATE ACCEPTANCE OF THIS AGREEMENT AND THE LIMITED WARRANTY AND LIMITATION OF LIABILITY SET OUT IN THIS AGREEMENT. YOU SHOULD THEREFORE READ THIS AGREEMENT CAREFULLY BEFORE INSTALLING, COPYING, DOWNLOADING, ACCESSING, OR OTHERWISE USING THIS SOFTWARE.

A manually signed license agreement between the Licensee and the Licensor, if applicable, will supersede any conflicting terms in this Agreement. The text of this Agreement can also be found in the on-line help system and printed from there.

LICENSE AGREEMENT

1. Evaluation

   If you have received a copy of the Software from Licensor or an authorized vendor, but you have not yet purchased a license to use the Software, the Licensor grants you a personal, non-transferable, non-exclusive, limited license to install and use the Software for your own internal use solely for purposes of evaluating the Software for no more than thirty (30) days. When the Software is used on an evaluation basis, it may not have the full functionality described in its accompanying documentation.

2. Grant of License and Permitted Uses

   a. Grant of License. Subject to the terms and conditions of this Agreement and any applicable terms and conditions of the purchase order or other agreement between the Licensee and Licensor that define the terms of the purchase and permitted use

of the Software, which terms and conditions are incorporated herein by reference, Licensor grants Licensee a limited, non-exclusive, non-transferable, license:

(i) for each AUTHORIZED DEVELOPER CPU COPY of the Software paid for by the Licensee, to install, operate and use each AUTHORIZED DEVELOPER CPU COPY of the Software solely for use by the authorized project team specified in the order form the Software; and

(ii) for each AUTHORIZED RUNTIME COPY of the Software paid for by the Licensee, to install, operate, and use such runtime copy for the term of this Agreement or to distribute to an end-user ("End-User") that agrees to be bound by the terms of this Agreement such runtime copy as an integral part of, and solely in conjunction with, a product or software program distributed by Licensee ("Developer Product") that is registered and approved by Licensor and not as a separate, unbundled product (except as needed to replace a customer's defective media, or to remedy an error in the Software that can be resolved by providing customer with an upgrade).

b. Copies. The order confirmation that was delivered with this product specifies the number of AUTHORIZED DEVELOPER CPU COPIES and AUTHORIZED RUNTIME COPIES of the Software that are permitted to Licensee. Except as otherwise expressly provided in this Agreement, Licensee may not copy the Software or any portion thereof, in whole or in part, except as is necessary to load, operate, use, and/or distribute the number of authorized copies of the Software specified in the applicable order confirmation. AUTHORIZED RUNTIME COPIES of the Software may be produced by Licensee from Developer CPUs or may be provided directly to Licensee by Licensor and must be free of any Software source code. For each authorized copy, the Licensee may make one (1) copy of the Software and the system configuration and other installation-specific files that are created during the installation and configuration process using the Software solely for archival purposes, provided that Licensee reproduces on the back-up copy all copyright notices and any other proprietary legends that are on or encoded in the Software. Licensee may transfer the Software and set-up and other installation-specific files from one server to another at no additional license fee provided that Licensee deletes the Software and configuration and other installation-specific files from the server no longer in use and from each back-up copy for that server.

c. Reservation of Rights. Any rights not expressly granted in this Agreement are reserved to Licensor.

3. Intellectual Property Restrictions and Other Limitations

a. Restrictions on Copying and Modifying. Except as otherwise expressly provided in this Agreement, Licensee shall not (i) copy the Software, in whole or in part; or (ii) adapt, alter, create derivative works based on, modify, or translate the Software, in whole or in part.

b. Open Source. Licensee shall not utilize the Software in conjunction with any Public Software in a manner which would require the Software to be disclosed or distributed in source code form or made available at no charge. "Public Software" means any software that contains, or is derived in any manner (in whole or in part) from, any software that is distributed as free software, open source software (e.g., Linux) or similar licensing or distribution models.

c. Restrictions on Transfer. Licensee may not (i) sell, assign, distribute, lease, market, rent, lend, sublicense, transfer, make available, or otherwise grant rights to the Software, in whole or in part, to any third party in any form; or (ii) electronically transfer the Software, in whole or in part, from one computer to another over a network except as is necessary to load, operate and use one installation copy of the Software.

d. Intellectual Property Notices and Marking. Licensee may not (i) obscure, remove or alter any of the trademarks, trade names, logos, patent or copyright notices or markings applied to or on the Software; or (ii) add any other notices or markings to the Software or any portion thereof.

e. Limitations on Reverse Engineer, Decompilation, and Disassembly. Licensee may not reverse engineer, decompile, or disassemble the Software or any portion thereof or otherwise obtain or attempt to obtain the source code for the Software or any portion thereof. If Software is provided with source code, Licensee acknowledges that the source code is confidential to Licensor. Licensee shall preserve the confidentiality of the source code and ensure that the source code is not disclosed, distributed, or available to third parties. Licensee shall limit access to the source code to the single Licensed user as specified in the order confirmation.

f. Restrictions on Separation of Components. The Software is licensed as a single product. Licensee may not separate or attempt to separate any of the components of the Software. The component parts of the Software may not be separated for use on more than one computer.

4. Support for Software

Product support for the Software is not provided by Licensor unless by some specific support agreement.

5. Ownership of Software

Licensor has and shall have sole and exclusive ownership of all right, title, and interest in and to the Software and all portions and copies thereof. In addition, Licensor shall have sole and exclusive ownership of any additional programs, updates, modifications, fixes, patches, upgrades, and revisions provided to Licensee for the Software. No title is transferred by this Agreement or by the payment of any fee. If title to the Software or any part or element thereof does not, by operation of law, vest in Licensor, Licensee hereby assigns to Licensor, or its designee all right, title and interest in and to the Software.

6. Fees

In partial consideration for the rights granted to Licensee herein, Licensee shall pay to Licensor or an authorized vendor the fees in accordance with the terms and conditions of the purchase order or other agreement(s) between the Licensee and Licensor or an authorized vendor that define the purchase and permitted use of the Software, which terms and conditions are incorporated herein by reference. Each party will be responsible for its own expenses incurred in rendering performance under this Agreement, including the cost of facilities, work space, computers and computer time, development tools and platforms, utilities management, personnel, supplies and the like.

7. Failure to Execute Agreement

If you are unwilling or unauthorized to execute this Software License Agreement, you should destroy the Software and all copies thereof and notify Licensor or an authorized vendor from which it was received within ten (10) days thereof.

8. Term and Termination

a. Term. This Agreement commences upon the earliest date that you install, copy, download, use the Software or otherwise accept the terms and conditions of this Agreement, provided that the terms and conditions of the purchase order or other agreement between the Licensee and Licensor or an authorized vendor defining the purchase and permitted use of the Software have been satisfied, and shall continue until terminated as provided herein.

b. Termination. Licensee may terminate this Agreement at any time, with or without cause, by returning to Licensor or destroying the Software and all copies thereof and deleting or uninstalling the Software and all copies thereof, and certifying the same in writing to Licensor within ten (10) business days of termination.

c. Effect of Termination. Termination of this Agreement shall not relieve either party of any obligation or liability accrued hereunder prior to such termination, nor affect or impair the rights of either party arising under this Agreement prior to such termination, except as expressly provided herein. Upon termination, the Licensee agrees to promptly return to Licensor or destroy the Software and all copies thereof and delete or uninstall the Software and all copies thereof, and certify the same in writing to Licensor within ten (10) business days of termination.

9. Disclaimers and Remedies.

a. Disclaimer of Warranty. THE SOFTWARE AND ALL PORTIONS THEREOF, AND ANY SERVICES ARE PROVIDED “AS IS.” TO THE MAXIMUM EXTENT PERMITTED BY LAW, LICENSOR DISCLAIMS ALL OTHER WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, PERFORMANCE, AND FITNESS FOR A PARTICULAR PURPOSE. LICENSOR DOES NOT WARRANT THAT THE FUNCTIONS OR INFORMATION CONTAINED IN THE SOFTWARE WILL MEET ANY REQUIREMENTS OR NEEDS LICENSEE MAY HAVE, OR THAT THE Software WILL OPERATE ERROR FREE, OR IN AN UNINTERRUPTED FASHION, OR THAT ANY DEFECTS OR ERRORS IN THE SOFTWARE WILL BE CORRECTED, OR THAT THE SOFTWARE IS COMPATIBLE WITH ANY PARTICULAR SYSTEM, NETWORK, OR SOFTWARE.

b. Limitations of Liability. TO THE MAXIMUM EXTENT PERMITTED BY LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO LICENSEE OR ANY THIRD PARTY FOR ANY LOSS OF PROFITS, LOSS OF USE, LOSS OF DATA, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, INDIRECT, SPECIAL, PUNITIVE, OR EXEMPLARY DAMAGES) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR ANY PORTION THEREOF, DEFECTS IN WARRANTY, ANY SERVICES, OR FOR ANY CLAIM BY ANY OTHER PARTY, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL LICENSOR’S LIABILITY EXCEED THE AMOUNT OF FEES PAID UNDER THIS AGREEMENT (WHETHER SUCH LIABILITY ARISES FROM BREACH OF WARRANTY, BREACH OF THIS AGREEMENT, OR OTHERWISE, AND WHETHER IN CONTRACT OR IN TORT,

INCLUDING NEGLIGENCE AND STRICT LIABILITY). IN NO EVENT MAY ANY ACTION BE BROUGHT AGAINST LICENSOR ARISING OUT OF THIS AGREEMENT MORE THAN ONE YEAR AFTER THE CLAIM OR CAUSE OF ACTION ARISES, DETERMINED WITHOUT REGARD TO WHEN THE LICENSEE SHALL HAVE LEARNED OF THE ALLEGED DEFECT, INJURY, OR LOSS. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO EACH LICENSEE. THE PROVISIONS OF THIS SECTION 9 WILL SURVIVE ANY TERMINATION OF THIS AGREEMENT.

10. Security and Audit Rights

Licensee will take reasonable action to restrict and control all use of the Software to enforce the limitations and restrictions set forth in Sections 2 and 3 of this Agreement, and to permit access only to End-Users authorized to use the Software. Licensee will make reasonable efforts to ensure compliance by all End-Users authorized to use the Software with this Agreement. Licensee grants Licensor the right to audit, at any time during regular business hours without prior notice, use of the Software to ensure compliance with this Agreement.

11. Indemnification.

a. Licensor. If Licensee receives a claim that the use of the Software infringes a patent, copyright, or other Intellectual property right, Licensee must promptly notify Licensor in writing. Licensor shall, at its own expense and option: (i) defend and settle such claim, (ii) procure Licensee the right to use the Software, (iii) modify or replace the Software to avoid infringement; or (iv) provide a pro rata refund of license fees paid for the applicable time period. In the event Licensor exercises option (i) above, it shall have the sole and exclusive authority to defend and/or settle any such claim or action.

b. Licensee. Licensee agrees to indemnify, defend, and hold harmless Licensor and its directors, officers, employees, agents, successors and assigns from and against any and all third-party liabilities, claims, demands losses, damages, costs and expenses (including reasonable attorney's fees) which may be assessed against or incurred by Licensee relating to or arising out of: (i) any material breach of this Agreement by Licensee; (ii) any allegation that one or more of the Developer Products infringes the intellectual property rights of a third party; (iii) the use of the Client Libraries in a manner prohibited under this Agreement, or in a manner for which the Client Libraries were not designed; or (iv) any negligent, grossly negligent or intentional misconduct or omission of Licensee or its directors, officers, employees, agents, successors and assigns in connection with its use of the Software.

c. Exceptions. Licensor shall have no liability to Licensee under section 11(a) or otherwise for any claim or action alleging infringement based upon (i) any use of the Software in a manner other than as specified by Licensor; (ii) any combination of the Software by Licensee with other products, equipment, devices, software, systems, or data not supplied by an authorized vendor or Licensor (including, without limitation, any software produced by Licensee for use with the Software or Developer Products) to the extent such claim is directed against such combination; (iii) any unauthorized modifications, enhancements or customization of the Software by any person other than Licensor; or (iv) use of other than a current release of the Software, if such infringement would have been avoided by use of a current release that Licensor has made available to Licensee free of charge prior to the notice of infringement.

12. General Provisions

a. Severability. Should any term or provision of this Agreement be finally determined by a court of competent jurisdiction to be void, invalid, unenforceable or contrary to law or equity, the offending term or provision shall be modified and limited (or if strictly necessary, deleted) only to the extent required to conform to the requirements of law and the remainder of this Agreement (or, as the case may be, the application of such provisions to other circumstances) shall not be affected thereby but rather shall be enforced to the greatest extent permitted by law, and the parties shall use their best efforts to substitute for the offending provision new terms having similar economic effect.

b. Governing Law. This Agreement shall for all purposes be governed by and interpreted in accordance with the laws of the Commonwealth of Massachusetts, as those laws are applied to contracts entered into and to be performed entirely in Massachusetts, without reference to its conflicts of laws provisions. Any legal suit, action, or proceeding arising out of or relating to this Agreement shall be commenced in a federal or state court in Massachusetts, and each party hereto irrevocably submits to the non-exclusive jurisdiction and venue of any such court in any such suit, action, or proceeding. Licensee hereby acknowledges and agrees that the U.N. Convention on Contracts for the International Sale of Goods shall not apply to this Agreement.

c. U.S. Government Rights. If you are a branch or agency of the U.S. Government, then you acknowledge that the Software is a "commercial item" as that term is defined at 48 C.F.R. 2.101, consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212. Any technical data provided with the Software is commercial technical data as defined in 48 C.F.R. 12.211. Consistent with 48 C.F.R. 12.211 through 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4, and 48 C.F.R. 252.227-7015, all U.S. Government end users acquire the Software with only the rights set forth in this Agreement.

d. Modification and Waiver. Any modification, amendment, supplement, or other change to this Agreement must be in writing and signed by a duly authorized representative of Licensor and Licensee. All waivers must be in writing. The failure of Licensor to insist upon strict performance of any provision of this Agreement, or to exercise any right provided for herein, shall not be deemed to be a waiver of the future performance or exercise of such provision or right, and no waiver of any provision or right shall affect the right of the waiving party to enforce any other provision or right herein.

e. Assignment. No right or obligation of Licensee under this Agreement may be assigned, delegated or otherwise transferred, whether by agreement, operation of law, or otherwise, without the express prior written consent of Licensor, and any attempt to assign, delegate or otherwise transfer any of Licensee's rights or obligations hereunder, without such consent, shall be void. Subject to the preceding sentence, this Agreement shall bind each party and its permitted successors and assigns.

f. Remedies. The parties agree that any breach of this Agreement would cause irreparable injury for which no adequate remedy at law exists; therefore, the parties agree that equitable remedies, including without limitation, injunctive relief and specific performance, are appropriate remedies to redress any breach or threatened breach of this Agreement, in addition to other remedies available to the parties. All rights and

remedies hereunder shall be cumulative, may be exercised singularly or concurrently, and shall not be deemed exclusive except as otherwise provided. If any legal action is brought to enforce any obligations hereunder, the prevailing party shall be entitled to receive its attorneys' fees, court costs, and other collection expenses, in addition to any other relief it may receive. Licensee hereby waives any right or claim to which Licensee may be entitled to immunity or exemption from liability.

g. Notice. All notices, statements and reports required or permitted by this Agreement shall be in writing and deemed to have been effectively given and received: (i) on the date shown on the return receipt if sent by registered or certified U.S. Mail, postage prepaid, with return receipt requested; or (ii) when delivered if delivered personally or sent by express courier service provided a receipt of delivery is obtained. Notices shall be addressed as follows:

PC Phone Connections
Email: sales@pcphoneconnections.com

h. Force Majeure. Neither party will be responsible for any failure to fulfill its obligations due to causes beyond its reasonable control, including without limitation, acts or omissions of government or military authority, acts of God, materials shortages, transportation delays, fires, floods, labor disturbances, riots, wars, or inability to obtain any export or import license or other approval of authorization of any government authority.

i. Export Control. Licensee shall not export or allow the export or re-export of the Software or any portions thereof without compliance with all export laws and regulations of the U.S. Department of Commerce and all other U.S. agencies and authorities, including without limitation, the Export Administration Regulations of the U.S. Department of Commerce, Bureau of Export Administration, and, if applicable, relevant foreign laws and regulations.

j. Relationship. Licensor and Licensee are independent contracting parties. This Agreement shall not constitute the parties as principal and agent, partners, joint venturers, or employer and employee.

k. Entire Agreement. This Agreement constitutes the entire, full and complete Agreement between the parties concerning the subject matter hereof, and they collectively supersede all prior or contemporaneous oral or written communications, proposals, conditions, representations and warranties. This Agreement prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter.

2. Introduction

The **EZTapiAPI SDK** is a Software Library written in the C/C++ programming language that was created to offer software developers an easy to use software interface for controlling Voice Modems in computers running Microsoft Windows Operating Systems. The **EZTapiAPI SDK** Library exports a set of C function calls, which make up the library's Application Programming Interface (API). These easy to use function calls replace the more complex operations that would be needed for building Interactive Voice Response Systems (IVR) with the Microsoft TAPI API directly.

The **EZTapiAPI SDK** offers both a synchronous and asynchronous interface for many of the exported APIs. This offers the developer the needed flexibility they may require to quickly add a telephony interface to new or existing applications.

The **EZTapiAPI SDK** also gives programmers a tool to develop telephony-based applications in a non-windows environment. All function calls can be made from traditional console based applications written in C/C++ when compiled with TAPI version 2.0 or greater. Under Windows 95, console applications are not supported.

Beginning with release version 2.0 of the **EZTapiAPI SDK**, all exported functions have been defined to use the __stdcall convention, allowing them to be called from Visual Basic applications. See the sample Visual Basic application for more information.

3. Hardware/Software Requirements

Although the **EZTapiAPI** library is designed to work in all windows based environments, your development system must include a release of the Microsoft TAPI API version 1.4 or greater. The **EZTapiAPI** library does not work with the COM based programming model of TAPI 3.0.

The **EZTapiAPI** library can communicate with most Voice Modems through the Unimodem V or 5 Telephony Service Provider (TSP) shipped with most Windows Operating Systems including Windows 95, Windows 98, Windows 2000, and Windows XP.

You must be using a voice modem to use many of the **EZTapiAPI** function calls. Some voice modems are equipped with an on-board voice resource. If your modem does not have an on-board voice resource, it will need to be connected to a sound card with a Telephone Answering Device (TAD) interface.

Note: Windows NT does not have support for the Unimodem TSP and therefore will not work with the **EZTapiAPI SDK**.

Windows 95 users may need to get updates from Microsoft Corporation in order to take advantage of the voice features available with the release of Unimodem V.

4. Overview of EZTapiAPI

The **EZTapiAPI** is a Software Library that exports a set of easy to use C functions for developing telephony applications that work with voice modems in computers running Microsoft Windows Operating Systems.

To use the **EZTapiAPI** interface, the developer must first initialize the **EZTapiAPI** engine. This is accomplished through the function **EZTapiInitialize**. After the **EZTapiAPI** engine has been initialized, a phone line must then be opened using the function **EZOpenLine**.

With the **EZTapiAPI** engine successfully initialized and a valid **EZLine** device successfully opened, the developer can then initiate outgoing calls with the function **EZMakeCall** or answer incoming calls using either **EZWaitForCall** or **EZAnswerCall**. Caller Id extraction is supported by the **EZTapiAPI SDK** for voice modems that support this feature. Caller Id is a subscriber call feature that must be provisioned on a phone line. This feature many times has an associated fee. For more information on Caller Id support the developer should contact their local telephone company.

Once an active call has been established over an **EZLine** device, the developer can then use any of the other exported **EZTapiAPI** functions for controlling the call. With these functions the developer is able to play and record wave file voice prompts, and collect and send DTMF and Pulse digits.

Most functions in the **EZTapiAPI** library can be called either synchronously or asynchronously. If the user wishes to make a function call asynchronously, the **EZTapiAPI** engine must be initialized with a valid pointer to a callback function that the user must define. Asynchronous results are posted back from the **EZTapiAPI** to the client application through this callback interface. For a list of **EZTapiAPI** Event messages see Appendix A.

For functions called synchronously, the user does not need to provide this callback interface. Function calls will block until the operation completes either successfully or in error. Appendix B contains a list of **EZTapiAPI** Errors and their meanings.

The user is free to make calls both synchronously and asynchronously within the same application. The figure below is a state chart diagram that helps to explain the **EZTapiAPI SDK** interface. This diagram shows the **EZLine** state transitions in response to **EZTapiAPI** function calls.

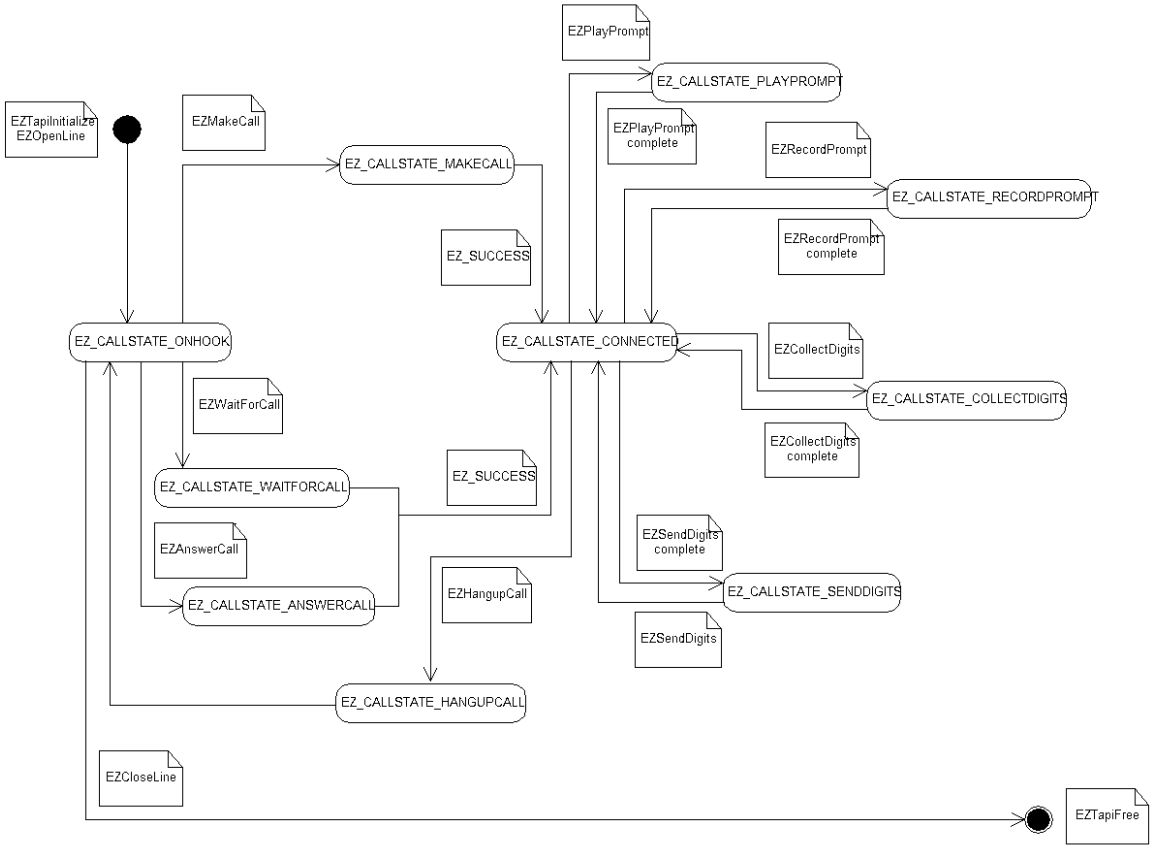

Figure 1. State Chart Diagram for the EZTapiAPI SDK.

5. Programming Model Example

The following sections discuss the ez_console sample application included with the **EZTapiAPI SDK**. This is a Win32 console application designed to demonstrate the **EZTapiAPI** library interface. This program is actually two demo applications in one.

Synchronous Programming Model

On startup of the ez_console application, the **EZTapiAPI** engine is initialized and a single **EZLine** device is opened. The **EZLine** device will be in the **EZ_CALLSTATE_ONHOOK** state after it has been successfully opened.

A menu is then presented to the user allowing the user to issue keyboard commands for placing or answering calls. Once a valid call has been established, the user can then issue keyboard commands to play, and record prompts, send and collect DTMF digits, and perform other call related tasks.

All **EZTapiAPI** function calls resulting from user input are made synchronously. **EZTapiAPI** functions are called synchronously by specifying an **EZCallType** value of **EZ_SYNC.**

Asynchronous Programming Model

As mentioned above, on startup of the ez_console application, the **EZTapiAPI** engine is initialized, and a single **EZLine** device is opened. The **EZLine** device will be in the **EZ_CALLSTATE_ONHOOK** state after it has been successfully opened.

Since a valid callback function pointer is specified in the call to **EZTapiInitialize**, the application is able to receive event notifications. If no user input in provided through the keyboard interface, this application will work as an answering machine after receiving four **EZ_RING_EVENTS**.

The application issues asynchronous **EZTapiAPI** function calls to answer the incoming call, play a voice prompt and then record a voice message. All functions are executed based on completion events received for the **EZLine** device. Completion events are posted for all functions called with an **EZCallType** value of **EZ_ASYNC**.

ez_console Demo Source Code

```
// System Include files
#include <stdio.h>
#include <windows.h>

// EZTapiAPI include files
#include "ezparams.h"
#include "ezerrors.h"
#include "eztapiapi.h"


enum
{
    TASK_MAKECALL,
    TASK_WAITFORCALL,
    TASK_ANSWERCALL,
    TASK_PLAYPROMPT,
    TASK_RECORDPROMPT,
    TASK_SENDDIGITS,
    TASK_COLLECTDIGITS,
    TASK_HANGUPCALL,
    TASK_GETCALLSTATE,
    TASK_QUIT,
    TASK_INVALIDENTRY
} CALL_TASK;


/*****************************************************************************
 Function: ProcessEZError()

 Description:  The following function simply identifies a particular error and
               prints the defined error in text form to standard out.

 Parameters:  error_num -- the numeric error identifier

 Return Values:  <none>
*****************************************************************************/
void ProcessEZError(DWORD error_num)
{
    switch (error_num)
    {
        case EZ_EVENTMGMT_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_EVENTMGMT_ERROR.\n", error_num);
            break;

        case EZ_HANGUP_ERROR:

           printf("ERROR: EZ_ERROR: %d == EZ_HANGUP_ERROR.\n", error_num);
            break;

        case EZ_BADFORMAT_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_BADFORMAT_ERROR.\n", error_num);
            break;

        case EZ_FILENOTFOUND_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_FILENOTFOUND_ERROR.\n", error_num);
            break;

        case EZ_BEEP_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_BEEP_ERROR.\n", error_num);
            break;

        case EZ_PARAM_ERROR:
```

```
            printf("ERROR: EZ_ERROR: %d == EZ_PARAM_ERROR.\n", error_num);
            break;

        case EZ_MEMORY_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_MEMORY_ERROR.\n", error_num);
            break;

        case EZ_RESOURCE_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_RESOURCE_ERROR.\n", error_num);
            break;

        case EZ_TIMEOUT_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_TIMEOUT_ERROR.\n", error_num);
            break;

        case EZ_NOTSP_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_NOTSP_ERROR.\n", error_num);
            break;

        case EZ_CALLBUSY_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_CALLBUSY_ERROR.\n", error_num);
            break;

        case EZ_CALLNOANSWER_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_CALLNOANSWER_ERROR.\n", error_num);
            break;

        case EZ_CALLSTATE_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_CALLSTATE_ERROR.\n", error_num);
            break;

        case EZ_TAPIVERSION_ERROR:

            printf("ERROR: EZ_ERROR: %d == EZ_TAPIVERSION_ERROR.\n", error_num);
            break;

        default:

            printf("ERROR: Unknown EZ_ERROR: %d.\n", error_num);
            break;

    } // end switch()

    return;
}


/*****************************************************************************
 Function:  ez_callback()

 Description:  EZTapiAPI callback function, which is needed to handle
               asynchronous EZTapiAPI function calls.  The EZTapiAPI engine
               signals EZLine events and errors back to the client application
               via this mechanism.

 Parameters:  aLine -- line on which an event occurred
              EZEvent -- event which caused callback function to be called.
              ezdata -- pointer to some user data associated with EZEvent
              usrData -- user defined data pointer

 Return Values: <none>
*****************************************************************************/
```

```
void ez_callback(EZLine aLine, EZ_EVENT EZEvent, VOID *ezdata, VOID *usrData)
{
    DWORD rc;
    DWORD *appData;
    static DWORD tot_rings = 0;
    EZCallInfo *pCallInfo = NULL;
    char szDigits[EZ_DIGITBUFFERSIZE];
    char szFileName[EZ_MAXPATHSIZE];
    EZWaveFormat waveFmt;

    /*********************************************************************
     Cast the VOID pointer usrData, to its appropriate type.  Here simply
     casting to a DWORD pointer, then checking to make sure value was
     received correctly.
    *********************************************************************/
    appData = (DWORD *) usrData;
    if (*appData != -999)
        printf("ERROR: ez_callback called with incorrect appData value.\n");


    switch (EZEvent)
    {
        case EZ_RING_EVENT:

            printf("INFO: ez_callback received EZ_RING_EVENT.\n");

            tot_rings += 1;

            if (tot_rings == 4)
            {
                /*************************************************************
                 Phoneline has received 4 rings.  Assume no one is available to
                 answer the call.  Answer the call to take a message.
                *************************************************************/
                printf("INFO: Calling EZAnswerCall().\n");

                rc = EZAnswerCall(aLine, EZ_ASYNC);
                if (rc != EZ_SUCCESS)
                {
                    printf("ERROR: An error occurred calling EZAnswerCall().\n");
                    ProcessEZError(rc);
                }

            } // if (tot_rings == 4)

            break;

        case EZ_CALLCONNECT_EVENT:

            printf("INFO: ez_callback received EZ_CALLCONNECT_EVENT.\n");
            break;

        case EZ_CALLBUSY_EVENT:

            printf("INFO: ez_callback received EZ_CALLBUSY_EVENT.\n");
            break;

        case EZ_CALLERID_EVENT:

            printf("INFO: ez_callback received EZ_CALLERID_EVENT.\n");

            /*****************************************************************
             Cast the VOID *ezdata value to a EZCallInfo structure to retrieve
             Caller ID information.
            *****************************************************************/
            pCallInfo = (EZCallInfo *) ezdata;
            printf("INFO: Caller Name: %s, Caller Number: %s.\n",
                   pCallInfo->CallerName, pCallInfo->CallerNumber);
```

```
        break;

    case EZ_CALLANSWER_EVENT:

        printf("INFO: ez_callback received EZ_CALLANSWER_EVENT.\n");

        tot_rings = 0;

        printf("INFO: Calling EZPlayPrompt().\n");

        rc = EZPlayPrompt(aLine, "nobodyhome.wav", EZ_ASYNC);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZPlayPrompt().\n");
            ProcessEZError(rc);
        }

        break;

    case EZ_CALLHANGUP_EVENT:

        printf("INFO: ez_callback received EZ_CALLHANGUP_EVENT.\n");

        tot_rings = 0;

        printf("INFO: Calling EZHangupCall().\n");

        rc = EZHangupCall(aLine);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZHangupCall().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZHangupCall() successful.\n\n");
        }

        break;

    case EZ_PLAYBEGIN_EVENT:

        printf("INFO: ez_callback received EZ_PLAYBEGIN_EVENT.\n");
        break;

    case EZ_PLAYSTOP_EVENT:
    case EZ_PLAYEND_EVENT:

        if (EZEvent == EZ_PLAYSTOP_EVENT)
            printf("INFO: ez_callback received EZ_PLAYSTOP_EVENT.\n");
        else
            printf("INFO: ez_callback received EZ_PLAYEND_EVENT.\n");


        /*****************************************************************
         Set EZWaveFormat to record an 8000Khz, 16 bits/sample, mono
         wave file recording.
        *****************************************************************/
        waveFmt.NumChannels = 1;
        waveFmt.SampleRate = 8000;
        waveFmt.BitsPerSample = 16;

        printf("INFO: Calling EZRecordPrompt().\n");

        /*****************************************************************
         Calling EZRecordPrompt() with empty szFileName.  EZTapi will
         create a unique filename from the computer date and time.
        *****************************************************************/
        rc = EZRecordPrompt(aLine, szFileName, &waveFmt, true, EZ_ASYNC);
```

```
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZRecordPrompt().\n");
            ProcessEZError(rc);
        }

        break;

    case EZ_RECORDBEGIN_EVENT:

        printf("INFO: ez_callback received EZ_RECORDBEGIN_EVENT.\n");
        break;

    case EZ_RECORDSTOP_EVENT:
    case EZ_RECORDEND_EVENT:

        if (EZEvent == EZ_RECORDSTOP_EVENT)
            printf("INFO: ez_callback received EZ_RECORDSTOP_EVENT.\n");
        else
            printf("INFO: ez_callback received EZ_RECORDEND_EVENT.\n");

        /*****************************************************************
         Cast parameter 3 of ez_callback() to get name of recording file.
        *****************************************************************/
        strcpy(szFileName, (char *) ezdata);
        printf("INFO: New wave file recording: %s.\n", szFileName);

        break;

    case EZ_COLLECTDIGITBEGIN_EVENT:

        printf("INFO: ez_callback received EZ_COLLECTDIGITBEGIN_EVENT.\n");
        break;

    case EZ_COLLECTDIGITTIMEOUT_EVENT:
    case EZ_COLLECTDIGITSTOP_EVENT:
    case EZ_COLLECTDIGITEND_EVENT:

        if (EZEvent == EZ_COLLECTDIGITTIMEOUT_EVENT)
            printf("INFO: ez_callback received EZ_COLLECTDIGITTIMEOUT_EVENT.\n");
        else if (EZEvent == EZ_COLLECTDIGITSTOP_EVENT)
            printf("INFO: ez_callback received EZ_COLLECTDIGITSTOP_EVENT.\n");
        else
            printf("INFO: ez_callback received EZ_COLLECTDIGITEND_EVENT.\n");

        /*****************************************************************
         Cast parameter 3 of ez_callback() to get any collected digits.
        *****************************************************************/
        strcpy(szDigits, (char *) ezdata);
        printf("INFO: EZCollectDigits() done.  Received Digit String: %s\n",
               szDigits);

        printf("INFO: Calling EZFlushDigitBuffer().\n");

        rc = EZFlushDigitBuffer(aLine);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZFlushDigitBuffer.\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZFlushDigitBuffer() completed successfully.\n");
}

        break;

    case EZ_SENDDIGITBEGIN_EVENT:
```

```
            printf("INFO: ez_callback received EZ_SENDDIGITBEGIN_EVENT.\n");
            break;

        case EZ_SENDDIGITSTOP_EVENT:

            printf("INFO: ez_callback recevied EZ_SENDDIGITSTOP_EVENT.\n");
            break;

        case EZ_SENDDIGITEND_EVENT:

            printf("INFO: ez_callback recevied EZ_SENDDIGITEND_EVENT.\n");
            break;

        case EZ_UNEXPECTED_EVENT:

            printf("WARNING: ez_callback received EZ_UNEXPECTED_EVENT.\n");
            break;

        default:

            printf("WARNING: ez_callback received unknown event.  Unknown Event: 0x%x\n",
                   EZEvent);
            break;

    } // end switch()

    return;

}// end ez_callback()


/*****************************************************************************
 Function: GetNextTask()

 Description:  The following function processes user input to decide the next call
task to execute.

 Parameters: <none>

 Return Value:  CALL_TASK constant
*****************************************************************************/
DWORD GetNextTask()
{
    char inputdata;
    DWORD CallTask;


    printf("\nINFO: Enter next call task to be performed.\n");

    inputdata = getchar();

    switch (inputdata)
    {
        case 'M':
        case 'm':

            CallTask = TASK_MAKECALL;
            break;

        case 'W':
        case 'w':

            CallTask = TASK_WAITFORCALL;
            break;

        case 'A':
        case 'a':

            CallTask = TASK_ANSWERCALL;
```

```
            break;

        case 'H':
        case 'h':

            CallTask = TASK_HANGUPCALL;
            break;

        case 'P':
        case 'p':

            CallTask = TASK_PLAYPROMPT;
            break;

        case 'R':
        case 'r':

            CallTask = TASK_RECORDPROMPT;
            break;

        case 'S':
        case 's':

            CallTask = TASK_SENDDIGITS;
            break;

        case 'C':
        case 'c':

            CallTask = TASK_COLLECTDIGITS;
            break;

        case 'G':
        case 'g':

            CallTask = TASK_GETCALLSTATE;
            break;

        case 'U':
        case 'u':

            CallTask = TASK_UPDATEEZPARAM;
            break;

        case 'Q':
        case 'q':

            CallTask = TASK_QUIT;
            break;

        default:

            printf("\n\n\nERROR: Invalid menu selection.\n\n");

            printf("Enter:\n\n");
            printf("  'M' or 'm' for TASK_MAKECALL.\n");
            printf("  'W' or 'w' for TASK_WAITFORCALL.\n");
            printf("  'H' or 'h' for TASK_HANGUPCALL.\n");
            printf("  'P' or 'p' for TASK_PLAYPROMPT.\n");
            printf("  'R' or 'r' for TASK_RECORDPROMPT.\n");
            printf("  'S' or 's' for TASK_SENDDIGITS.\n");
            printf("  'C' or 'c' for TASK_COLLECTDIGITS.\n");
            printf("  'U' or 'u' for TASK_UPDATEEZPARAM.\n");
            printf("  'Q' or 'q' for TASK_QUIT.\n\n");

            CallTask = TASK_INVALIDENTRY;
            break;

    } // end switch()
```

```
    // Remove trailing 'LF' character.
    inputdata = getchar();

    return(CallTask);
}


/*****************************************************************************
 Function:  main()

 Description:  Entry point for the EZTapiAPI driver application.  Simple demo
               of EZTapiAPI functionality.

 Parameters:  <none>

 Return Values:  <none>
*****************************************************************************/
void main()
{
    DWORD rc;
    DWORD dwNumEZLineDev;
    EZModemType ModemType;
    EZCallInfo callInfo;
    char szFileName[EZ_MAXPATHSIZE], szDigits[EZ_DIGITBUFFERSIZE];
    EZWaveFormat waveFmt;
    EZLine aLine;
    DWORD CurrCallTask;
    DWORD userInfo = -999;
    DWORD i;
    DWORD dwCallState;


    printf("INFO: Calling EZInitializeTapi().\n");

    /*************************************************************************
     Initialize the EZTapiAPI Engine.  This must be the first call by a client
     application to the EZTapiAPI library.
    *************************************************************************/
    rc = EZTapiInitialize(EZ_CONSOLE_APP, &dwNumEZLineDev,
                          (EZ_CALLBACK) &ez_callback,
                          (void *) &userInfo);
    if (rc != EZ_SUCCESS)
    {
        printf("ERROR: An error occurred calling EZInitializeTapi().  EZTapiAPIError:
               %d.\n", rc);
        exit(1);
    }
    else
    {
        printf("INFO: Calling EZOpenLine().\n");

        /*********************************************************************
         Open the first EZLine device in the system.
        *********************************************************************/
        aLine = 0;

        rc = EZOpenLine(aLine, EZ_OWNER, &ModemType);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZOpenLine().\n");
            ProcessError(rc);

            goto EZTAPI_FREE;
        }

        printf("INFO: Line %d, successfully opened.\n", aLine);

        printf("Enter:\n\n");
```

```
printf("  'M' or 'm' for TASK_MAKECALL.\n");
printf("  'W' or 'w' for TASK_WAITFORCALL.\n");
printf("  'H' or 'h' for TASK_HANGUPCALL.\n");
printf("  'P' or 'p' for TASK_PLAYPROMPT.\n");
printf("  'R' or 'r' for TASK_RECORDPROMPT.\n");
printf("  'S' or 's' for TASK_SENDDIGITS.\n");
printf("  'C' or 'c' for TASK_COLLECTDIGITS.\n");
printf("  'Q' or 'q' for TASK_QUIT.\n\n");


while (true)
{
    CurrCallTask = GetNextTask();

    switch (CurrCallTask)
    {
        case TASK_MAKECALL:

            printf("INFO: Calling EZMakeCall().\n");

            rc = EZMakeCall(aLine, "103", EZ_SYNC);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling EZMakeCall().\n");
                ProcessEZError(rc);
            }

            break;

        case TASK_WAITFORCALL:

            printf("INFO: Calling EZWaitForCall().\n");

            rc = EZWaitForCall(aLine, 1, &callInfo);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling EZWaitForCall().\n");
                ProcessError(rc);
            }
            break;

        case TASK_ANSWERCALL:

            printf("INFO: Calling EZAnswerCall().\n");

            rc = EZAnswerCall(aLine, EZ_SYNC);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling EZAnswerCall.\n");
                ProcessError(rc);
            }

            break;

        case TASK_PLAYPROMPT:

            printf("INFO: Calling EZPlayPrompt().\n");

            rc = EZPlayPrompt(aLine, "thankyou.wav", EZ_SYNC);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling EZPlayPrompt().\n");
                ProcessEZError(rc);
            }
            else
            {
                printf("INFO: EZPlayPrompt() successful.\n");
            }
```

```
        break;

    case TASK_RECORDPROMPT:

        printf("INFO: Calling EZRecordPrompt().\n");

        waveFmt.NumChannels = 1;
        waveFmt.SampleRate = 8000;
        waveFmt.BitsPerSample = 16;

        memset(szFileName, '\0', EZ_MAXPATHSIZE);

        rc = EZRecordPrompt(aLine, szFileName, sizeof(szFileName),
                            &waveFmt, true, EZ_SYNC);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZRecordPrompt().\n");
           ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZRecordPrompt() successful.\n");
            printf("INFO: New wave file recording: %s.\n", szFileName);
        }

        break;

    case TASK_SENDDIGITS:

        printf("INFO: Calling EZSendDigits().\n");

        rc = EZSendDigits(aLine, "!,102", EZ_SYNC);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZSendDigits().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZSendDigits() successful.\n");
        }

        break;

    case TASK_COLLECTDIGITS:

        memset(szDigits, '\0', EZ_DIGITBUFFERSIZE);

        rc = EZCollectDigits(aLine, szDigits, 4, "#", EZ_SYNC);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZCollectDigits().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZCollectDigits() successful.\n");
            printf("INFO: Digits collected: %s.\n", szDigits);
        }

        break;

    case TASK_HANGUPCALL:

        printf("INFO: Calling EZHangupCall().\n");

        rc = EZHangupCall(aLine);
        if (rc != EZ_SUCCESS)
        {
```

```
            printf("ERROR: An error occurred calling EZHangupCall().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZHangupCall() successful.\n\n");
        }

        break;

    case TASK_GETCALLSTATE:

        printf("INFO: Calling EZGetCallState().\n");
        printf("INFO: EZCallState: %d.\n", EZGetCallState(aLine));

        break;

    case TASK_UPDATEEZPARAM:

        if (toggle)
        {
            toggle = false;

            rc = EZSetParameter(EZ_DIGITWAVESTOP_PARAM, 0);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling
                       EZSetEZTapiAPIParam().\n");
                ProcessEZError(rc);
            }
            else
            {
                printf("INFO: EZ_DIGITSTOP_PARAM set to FALSE.\n");
            }

            rc = EZSetParameter(EZ_FIRSTDIGITTIMEOUT_PARAM, 20000);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling
                       EZSetEZTapiAPIParam().\n");
                ProcessEZError(rc);
            }
            else
            {
                printf("INFO: EZ_FIRSTDIGITTIMEOUT_PARAM set to 20000 (20
                       seconds).\n");
            }

            rc = EZSetParameter(EZ_INTERDIGITTIMEOUT_PARAM, 15000);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling
                       EZSetEZTapiAPIParam().\n");
                ProcessEZError(rc);
            }
            else
            {
                printf("INFO: EZ_INTERDIGITTIMEOUT_PARAM set to 15000 (15
                       seconds).\n");
            }

            rc = EZSetParameter(EZ_RECORDHANGUPTRIM_PARAM, 5500);
            if (rc != EZ_SUCCESS)
            {
                printf("ERROR: An error occurred calling
                       EZSetEZTapiAPIParam().\n");
                ProcessEZError(rc);
            }
            else
```

```
        {
            printf("INFO: EZ_RECORDHANGUPTRIM_PARAM, set to 5500 (5.5
                  seconds).\n");
        }

        rc = EZSetParameter(EZ_RECORDDIGITTRIM_PARAM, 1500);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling
                  EZSetEZTapiAPIParam().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZ_RECORDDIGITTRIM_PARAM, set to 1500 (1.5
                   seconds).\n");
        }
    }
    else // Reset EZ_PARAMETERS back to their default values
    {
        toggle = true;

        rc = EZSetParameter(EZ_DIGITWAVESTOP_PARAM, 1);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling
                  EZSetEZTapiAPIParam().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZ_DIGITSTOP_PARAM set to TRUE.\n");
        }

        rc = EZSetParameter(EZ_FIRSTDIGITTIMEOUT_PARAM, 10000);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling
                  EZSetEZTapiAPIParam().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZ_FIRSTDIGITTIMEOUT_PARAM set to 10000 (10
                  seconds).\n");
        }

        rc = EZSetParameter(EZ_INTERDIGITTIMEOUT_PARAM, 8000);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling
                  EZSetEZTapiAPIParam().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZ_INTERDIGITTIMEOUT_PARAM set to 8000 (8
                  seconds).\n");
        }

        rc = EZSetParameter(EZ_RECORDHANGUPTRIM_PARAM, 0);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling
                  EZSetEZTapiAPIParam().\n");
            ProcessEZError(rc);
        }
        else
        {
```

```
                            printf("INFO: EZ_RECORDHANGUPTRIM_PARAM, set to 0 (0
                                  seconds).\n");
                        }

                        rc = EZSetParameter(EZ_RECORDDIGITTRIM_PARAM, 0);
                        if (rc != EZ_SUCCESS)
                        {
                            printf("ERROR: An error occurred calling
                                  EZSetEZTapiAPIParam().\n");
                            ProcessEZError(rc);
                        }
                        else
                        {
                            printf("INFO: EZ_RECORDDIGITTRIM_PARAM, set to 0 (0
                                  seconds).\n");
                        }
                    }

                    break;

                case TASK_QUIT:

                    printf("INFO: Quitting Application.\n");
                    break;

                default:

                    break;

            } // end switch(CurrCallTask)


            if (CurrCallTask == TASK_QUIT)
                break;

        } // end while (true)

    } // end if (EZInitializeTapi())


    printf("\n\n");

    /*************************************************************************
     Get the current callstate of this line to see if this phoneline is still
     off-hook
    *************************************************************************/
    dwCallState = EZGetCallState(aLine);

    if (dwCallState == EZ_CALLSTATE_CONNECTED)
    {
        printf("INFO: Calling EZHangupCall().\n");

        rc = EZHangupCall(aLine);
        if (rc != EZ_SUCCESS)
        {
            printf("ERROR: An error occurred calling EZHangupCall().\n");
            ProcessEZError(rc);
        }
        else
        {
            printf("INFO: EZHangupCall() successful.\n\n");
        }
    }


EZTAPI_FREE:

    printf("INFO: Calling EZLineClose().\n");
```

```
    rc = EZCloseLine(aLine);
    if (rc != EZ_SUCCESS)
    {
        printf("ERROR: An error occurred calling EZLineClose().\n");
        ProcessEZError(rc);
    }
    else
    {
        printf("INFO: EZLineClose() successful.\n");
    }

    printf("INFO: Calling EZTapiFree().\n");

    rc = EZTapiFree();
    if (rc != EZ_SUCCESS)
    {
        printf("ERROR: An error occurred calling EZTapiFree().\n");
        ProcessEZError(rc);
    }
    else
    {
        printf("INFO: EZTapiFree successful.\n");
    }

    printf("INFO: Exiting application....\n");

    return;

} // end main()
```

6. EZTapiAPI Functions

6.1 Initialization/Resource Allocation Functions

LONG EZTapiInitialize(EZAppType *appType*, **DWORD** ***pdwNumEZLineDev**, **EZ_CALLBACK** ***pCallback**, **void** ***pUserInfo**)

Parameters

appType – Client application type. Should be set to **EZ_CONSOLE**, or **EZ_WINDOWS**.

pdwNumEZLineDev – Pointer to a **DWORD** that will be set to the number of discovered **EZLine** devices.

pCallback – Pointer to a user implemented callback function to receive asynchronous events from the **EZTapiAPI** engine. This parameter can be set to NULL if all **EZTapiAPI** functions are called with **EZCallType** set to **EZ_SYNC**.

pUserInfo – Void pointer to user defined data. Should be set to NULL if *pCallback* is NULL.

Function Description

EZTapiInitialize is called to initialize the **EZTapiAPI** engine. During initialization a search of the computer system is performed to discover all **EZLine** devices that can then be opened and controlled using the **EZTapiAPI SDK**.

The parameter *appType* should be set to **EZ_CONSOLE**, or **EZ_WINDOWS**. If the client application provides a Windows Message Pump for retrieving Windows Messages then *appType* can be set to **EZ_WINDOWS**. If the client application does not have a separate thread to process Windows Messages then **EZ_CONSOLE** should be specified. When specifying **EZ_CONSOLE**, the client application must be compiled with Microsoft's TAPI API version 2.0 or greater.

The second parameter *pdwNumEZLineDev* will be initialized by the **EZTapiAPI** engine and will hold the total number of **EZLine** devices discovered in the computer system. **EZLine** device ids are zero based. When calling other **EZTapiAPI** functions to open or otherwise control an **EZLine** device, the **EZLine** device id should be specified as a value from 0 to (*dwNumEZLineDev* – 1).

The third parameter *pCallback* should point to a user defined **EZ_CALLBACK** function that has been implemented in the client application. A valid pointer must be specified if any **EZTapiAPI** functions will be called asynchronously. This callback function receives completion events from the **EZTapiAPI** engine for **EZTapiAPI** functions that are called asynchronously. In addition to completion events for asynchronous function calls, the **EZTapiAPI** engine sends **EZ_EVENT** messages to signal dynamic events observed on open **EZLine** devices. If all other **EZTapiAPI** functions are called synchronously then this parameter may be set to NULL.

The last parameter to **EZTapiInitialize** is a pointer to user-defined data type. This value will be passed back to the client application as the last parameter of the user defined callback function *pCallback* when **EZ_EVENTS** are reported.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_MEMORY_ERROR**, **EZ_TSP_ERROR**, **EZ_TAPIVERSION_ERROR**, and **EZ_EVENTMGMT_ERROR**.

EZ_EVENT(s) Reported

none

LONG VBEZTapiInitialize(DWORD **pdwNumEZLineDev*, HWND *hWnd*)

Parameters

pdwNumEZLineDev – Pointer to a **DWORD** that will be set to the number of discovered **EZLine** devices.

hWnd – A handle to a VB Form. The **EZTapiAPI** engine will post **EZTapiAPI** event messages back to the VB Client application in response to asynchronous function calls. This parameter can be set to NULL if all **EZTapiAPI** functions are called with **EZCallType** set to **EZ_SYNC**.

Function Description

VBEZTapiInitialize is called to initialize the **EZTapiAPI** engine. During initialization a search of the computer system is performed to discover all **EZLine** devices that can then be opened and controlled using the **EZTapiAPI SDK**. This function should be called by Visual Basic client applications as an alternative to the C client initialization call **EZTapiInitialize.**

The parameter *pdwNumEZLineDev* will be initialized by the **EZTapiAPI SDK** and will hold the total number of **EZLine** devices discovered in the computer system. **EZLine** device ids are zero based. When calling other **EZTapiAPI** functions to open or otherwise control an **EZLine** device, the **EZLine** device id should be specified as a value from 0 to (*dwNumEZLineDev* – 1).

The parameter *hWnd* should be set to the Window Handle of an active VB Form. A valid window handle must be specified if any **EZTapiAPI** functions will be called asynchronously. Completion events for **EZTapiAPI** functions called asynchronously are reported back to the calling client application through this Window Handle interface. The **EZTapiAPI** engine will post messages back to the client application through this Window Handle interface, similar to the way native C applications receive completion events for asynchronous function calls through a user defined **EZ_CALLBACK** function *(See **EZTapiInitialize** function call above.)* The **EZTapiAPI** engine will also post **EZ_EVENT** messages to signal dynamic events observed on open **EZLine** devices. *(See the VB Sample application for more information on using the **EZTapiAPI** in asynchronous mode.)* This parameter can be set to NULL if all other **EZTapiAPI** functions are made as type **EZ_SYNC**.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_MEMORY_ERROR**, **EZ_TSP_ERROR**, **EZ_TAPIVERSION_ERROR**, and **EZ_EVENTMGMT_ERROR**.

EZ_EVENT(s) Reported

none

typedef VOID (__stdcall *EZ_CALLBACK) (EZLine *aLine*, **EZ_EVENT** *EZEvent*, **VOID *** *pEventData*, **VOID** *pUsrData***)**

Parameters

aLine – **EZLine** device on which the callback function is being signaled.

EZEvent – The **EZ_EVENT** that caused the callback function to be signaled.

pEventData – void pointer to additional data associated with the **EZ_EVENT**.

pUsrData – void pointer to user defined data.

Description

EZ_CALLBACK provides the definition for a callback function that needs to be defined and implemented by client **EZTapiAPI** application if other **EZTapiAPI** functions are to be called asynchronously.

The user defined and implemented **EZ_CALLBACK** will be called by the **EZTapiAPI** engine to report completion events for asynchronous function calls, as well as other **EZ_EVENT** messages to signal dynamic events observed on open **EZLine** devices. *(Visual Basic client applications will use the VBEZTapiInitialize function to initialize the EZTapiAPI engine and will not need to define an **EZ_CALLBACK** function.)*

The first parameter *aLine* is set to the **EZLine** device id on which the **EZ_EVENT** is being signaled.

The second parameter *EZEvent* is the actual **EZ_EVENT** that occurred on the **EZLine** device. If additional data is available for a particular **EZ_EVENT** it will be passed in the void pointer *pEventData*. For more information on **EZ_EVENT** values and associated data see Appendix A of this manual.

The value *pUsrData* is a pointer to any user-supplied data specified during initialization of the **EZTapiAPI** engine. It is the responsibility of the client application to make sure that this data does not go out of scope.

Return Values

none

EZ_EVENT(s) Reported

none

LONG EZTapiFree()

Parameters

none

Description

EZTapiFree is called to shut down the **EZTapiAPI** engine. This function closes all **EZLine** devices opened by the client application, and frees all **EZTapiAPI** resources. The client application should call this function before exiting.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**, otherwise the return value is negative. Possible return values include: **EZ_RESOURCE_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZOpenLine(EZLine *aLine*, EZCallPrivilege *callPrivilege*, EZModemType **pModemType*)

Parameters

aLine – **EZLine** device to be opened.

callPrivilege – **EZCallPrivilege** for the **EZLine** device. Should be set to **EZ_OWNER** or **EZ_MONITOR**.

pModemType – Pointer to an **EZModemType** value indicating if the modem was opened for voice or data operations. Will be set to either **EZ_MODEMTYPE_VOICE**, or **EZ_MODEMTYPE_DATA**.

Description

The function **EZLineOpen** is called to open an **EZLine** device associated with a modem device configured in a computer system. Prior to calling **EZLineOpen**, the client application must first initialize the **EZTapiAPI** engine to discover all **EZLine** devices in the computer system. **EZLine** device ids are zero based. *(See **EZTapiInitialize** for more information)*.

The parameter *callPrivilege* is used to specify the amount of control the application will have over the **EZLine** device being opened. If **EZLineOpen** is called with this parameter set to **EZ_OWNER**, the client application has complete control over the line and can issue further commands on that **EZLine** device. If **EZLineOpen** is called with *callPrivilege* set to **EZ_MONITOR**, the client application is not allowed direct control over the **EZLine** device and will merely receive status events for lines opened with this type of privilege.

The parameter *pModemType* will be initialized by the **EZTapiAPI** engine. It is used to indicate the type of operations allowed on the **EZLine** device. The **EZTapiAPI SDK** is a primarily a voice modem API, although limited support is available for data modems.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_TSP_ERROR**, **EZ_RESOURCE_ERROR**, and **EZ_PARAM_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZCloseLine(EZLine *aLine*)

Parameters

aLine – **EZLine** device to be closed.

Description

The function **EZCloseLine** is called to close an **EZLine** device associated with a modem device configured in a computer system. The **EZLine** device must have been previously opened with a call to **EZOpenLine**. Additionally the **EZLine** device must be in the **EZ_CALLSTATE_ONHOOK** state in order to be closed. **EZLine** device ids are zero based. *(See **EZTapiInitialize** for more information)*.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_CALLSTATE_ERROR**, and **EZ_RESOURCE_ERROR**.

EZ_EVENT(s) Reported

none

6.2Call Control/Setup Functions

LONG EZWaitForCall(EZLine *aLine*, DWORD *num_rings*, EZCallInfo **pcallInfo*)

Parameters

aLine – **EZLine** device on which an incoming call is to be answered.

num_rings – Number of rings to wait before answering the incoming call. This value must be greater than 0.

pCallInfo – Pointer to an **EZCallInfo** structure that will contain caller id information if available for the incoming call.

Description

The function **EZWaitForCall** is called to answer incoming phone calls on an **EZLine** device. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_ONHOOK** state. **EZWaitForCall** will block and only return after an incoming call has been successfully answered or some error occurred.

The parameter *num_rings* is used to specify how many rings the **EZLine** device should receive before answering the incoming call.

The third parameter *pCallInfo* will hold caller id information for the incoming call if this service is provisioned on the phone line and supported by the underlying modem hardware.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_CALLSTATE_ERROR**, **EZ_RESOURCE_ERROR**, and **EZ_EVENTMGMT_ERROR.**

EZ_EVENT(s) Reported

none

LONG EZAnswerCall(EZLine *aLine*, EZCallType *callType*)

Parameters

aLine – **EZLine** device on which an incoming call is to be answered.

callType – Function call type, should be set to **EZ_SYNC,** or **EZ_ASYNC**.

Description

The function **EZAnswerCall** is called to answer an incoming call on an **EZLine** device. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_ONHOOK** state.

Client applications would call this function in response to **EZ_RING_EVENT** messages being sent to a user defined **EZ_CALLBACK** function. The **EZTapiAPI** engine must be initialized with a valid pointer to a user defined **EZ_CALLBACK** function in order for the client application to be alerted to new incoming calls through **EZ_RING_EVENT** messages.

If **EZAnswerCall** is called with *callType* set to **EZ_SYNC**, the function will not return until it has completed and the call has been successfully answered or an error occurred. If **EZAnswerCall** is called with *callType* set to **EZ_ASYNC**, the function will return immediately to indicate whether or not the function was successfully started. The client application will be alerted to the overall success or failure of the function through the user defined **EZ_CALLBACK** function.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR, EZ_CALLSTATE_ERROR**, **EZ_HANGUP_ERROR**, **EZ_RESOURCE_ERROR**, and **EZ_EVENTMGMT_ERROR**.

EZ_EVENT(s) Reported

EZ_CALLANSWER_EVENT, and **EZ_UNEXPECTED_EVENT**

LONG EZMakeCall(EZLine *aLine*, char **szDialString*, EZCallType *callType*)

Parameters

aLine – **EZLine** device on which an outbound call is to be placed.

pszDialString – Pointer to a NULL terminated character string containing DTMF digits to be dialed.

callType – Function call type, should be set to **EZ_SYNC,** or **EZ_ASYNC**.

Description

The function **EZMakeCall** is called to initiate a new phone call over an opened **EZLine** device. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_ONHOOK** state.

The second parameter *pszDialString* is used to specify the phone number to be dialed.

If **EZMakeCall** is called with *callType* set to **EZ_SYNC**, the function will not return until it has completed and a new call has been successfully connected or an error occurred. If **EZMakeCall** is called with *callType* set to **EZ_ASYNC**, the function will return immediately to indicate whether or not the function was successfully started. The client application will be alerted to the overall success or failure of the function through the user defined **EZ_CALLBACK** function.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_CALLSTATE_ERROR,** **EZ_RESOURCE_ERROR**, **EZ_CALLNOANSWER_ERROR**, **EZ_CALLBUSY_ERROR**, and **EZ_EVENTMGMT_ERROR**.

EZ_EVENT(s) Reported

EZ_CALLCONNECT_EVENT, **EZ_CALLBUSY_EVENT**, **EZ_CALLNOANSWER_EVENT**, and **EZ_UNEXPECTED_EVENT**

Note: The Unimodem V/5 Telephony Service Providers (TSPs) do not report true call control information back to the **EZTapiAPI SDK**. This function may return prior to the call being actually connected. This is a limitation of this Unimodem TSPs.

LONG EZHangupCall(EZLine *aLine*)

Parameters

aLine – The **EZLine** device to be placed on-hook.

Description

The function **EZHangupCall** is called to put an **EZLine** device in the **EZ_CALLSTATE_ONHOOK** state. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_CONNECTED** state.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_CALLSTATE_ERROR, EZ_RESOURCE_ERROR**, and **EZ_EVENTMGMT_ERROR**.

EZ_EVENT(s) Reported

none

6.3Call Control/Processing Functions

LONG EZPlayPrompt(EZLine *aLine*, char **pszFileName*, EZCallType *callType*)

Parameters

aLine – **EZLine** device over which an audio prompt is to be played.

pszFileName – Pointer to a NULL terminated character string containing the name of the wave file to be played

callType – Function call type, should be set to **EZ_SYNC,** or **EZ_ASYNC**.

Description

The function **EZPlayPrompt** is called to initiate playback of a voice prompt over an **EZLine** device that is currently connected in an active call. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_CONNECTED** state.

The parameter *pszFileName* is used to specify the voice prompt to be played. This file should exist and be in wave file format.

If **EZPlayPrompt** is called with *callType* **EZ_SYNC**, the function will not return until playback has completed, playback is stopped, or some error has occurred. If **EZPlayPrompt** is called with *callType* set to **EZ_ASYNC**, the function will return immediately to indicate whether or not the function was successfully started. The client application will be alerted to the overall success or failure of the function through the user defined **EZ_CALLBACK** function.

By default, **EZPlayPrompt** will stop prompt playback when the **EZTapiAPI** engine recognizes a DTMF or Pulse digit. This skip-ahead feature is controlled by the **EZTapiAPI** system parameter **EZ_DIGITSTOP_PARAM**. Changing the value of this parameter via the **EZSetParameter** function can disable this behavior. See Appendix C for more information on **EZ_PARAMETER** system variables and their default settings.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_HANGUP_ERROR**, **EZ_CALLSTATE_ERROR, EZ_RESOURCE_ERROR**, **EZ_EVENTMGMT_ERROR**, **EZ_STOP_ERROR**, **EZ_FILENOTFOUND_ERROR**, **EZ_BADFORMAT_ERROR**, and **EZ_MEMORY_ERROR**.

EZ_EVENT(s) Reported

EZ_PLAYBEGIN_EVENT, **EZ_PLAYSTOP_EVENT**, **EZ_PLAYEND_EVENT**, and **EZ_UNEXPECTED_EVENT**

Note: If **EZPlayPrompt** returns the error **EZ_BADFORMAT_ERROR**, this can mean one of two things: either the wave file to be played is corrupt, and cannot be properly read by the EZTapiAPI library, or the modem is not capable of playback of wave audio data in the current wave format. If file specified by *filename* is known to be in valid wave file format, and the **EZ_BADFORMAT_ERROR** is returned, the developer should try converting the prompt to a different wave file format. Most voice modems will accept 8 kHz, Mono, 16 (or 8) bit wave files for playback and recording.

LONG EZRecordPrompt(EZLine *aLine*, **char ***_pszFileName_, **size_t** *bufferSize*, **EZWaveFormat ***_pWaveFmt_, **bool** *beep*, **EZCallType** *callType*)

Parameters

aLine – **EZLine** device over which the prompt is to be recorded.

pszFileName – Pointer to a NULL terminated character string containing the name that the wave file recording will be saved as. No .wav extension is required. May be set to NULL, or a NULL character string if **EZTapiAPI** engine is to generate a unique filename for the recording. *(See below description for more information on this feature.)*

bufferSize – The size of the character string buffer pointed to by *pszFileName*.

pWaveFmt – Pointer to an **EZWaveFormat** structure which defines the recording/sample rate information for wave recording.

beep – Boolean value which specifies whether a beep tone should be played before recording begins.

callType – Function call type, should be set to **EZ_SYNC,** or **EZ_ASYNC**.

Description

The function **EZRecordPrompt** is called to record PCM voice data over an **EZLine** device that is currently connected in an active call. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_CONNECTED** state.

The second parameter *pszFileName* is used to specify a name for the wave file recording. Although the developer does not need to include the ".wav" extension in the initial character string passed, the string buffer must be large enough to hold the four character ".wav" extension that will be appended to the character string pointer *pszFileName* by the **EZTapiAPI** engine.

If the client application specifies a pointer to a NULL terminated character string for the *filename* parameter, the **EZTapiAPI** engine saves the PCM voice data to a new unique file name based on the computer system data and time. The new file name will be copied to the filename pointer *pszFileName* and returned to the client application. If the client application wishes to pass a NULL terminated string to the **EZTapiAPI**, it is the responsibility of the client application to make sure that this character string is large enough to hold the file name created by the **EZTapiAPI** engine.

The parameter *bufferSize* should be set to the total size in memory of the character string buffer *pszFileName*.

The fourth parameter *pWaveFmt* is a pointer to an **EZWaveFormat** structure. This parameter is used to specify the audio format for the new wave recording. See Appendix D of this manual for more information on this structure.

The boolean parameter *beep* controls whether or not a short Beep Tone is to be played just before recording begins.

If **EZRecordPrompt** is called with *callType* **EZ_SYNC**, the function will not return until recording has completed, recording has been stopped, or some error has occured. If **EZRecordPrompt** is called with *callType* **EZ_ASYNC**, the function will return immediately to indicate whether or not the function was successfully started. The client application will be alerted to the overall success or failure of the function through the user defined **EZ_CALLBACK** function.

By default **EZRecordPrompt** will stop prompt recording when the **EZTapiAPI** engine recognizes a DTMF or Pulse digit. This skip-ahead feature is controlled by the **EZTapiAPI** system parameter **EZ_DIGITSTOP_PARAM**. This feature can be disabled using the **EZSetParameter** function. See Appendix C for more information on **EZ_PARAMETER** system variables and their default settings.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_HANGUP_ERROR**, **EZ_CALLSTATE_ERROR**, **EZ_RESOURCE_ERROR**, **EZ_BEEP_ERROR**, **EZ_STOP_ERROR**, **EZ_BADFORMAT_ERROR**, and **EZ_MEMORY_ERROR**.

EZ_EVENT(s) Reported

EZ_RECORDBEGIN_EVENT, **EZ_RECORDSTOP_EVENT**, **EZ_RECORDEND_EVENT**, and **EZ_UNEXPECTED_EVENT**

Note: If **EZRecordPrompt** returns the error **EZ_BADFORMAT_ERROR**, this can mean one of two things: either the **EZWaveFormat** specified is invalid, or the modem is not capable of recording wave audio data in the specified wave format. If the **EZWaveFormat** specified by pWaveFmt is found to be valid, and **EZ_BADFORMAT_ERROR** is returned, the developer should try changing the value of **EZWaveFormat** to a different sample rate. Most voice modems will accept 8 kHz, Mono, 16 (or 8) bit wave files for playback and recording.

LONG EZStopPrompt(EZLine *aLine*)

Parameters

aLine – **EZLine** device over which audio is being played or recorded.

Description

The function **EZStopPrompt** is called to stop playback or recording of wave file data. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_PLAYMESSAGE** or **EZ_CALLSTATE_RECORDMESSAGE** state.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_CALLSTATE_ERROR**, **EZ_RESOURCE_ERROR**, and **EZ_EVENTMGMT_ERROR**

EZ_EVENT(s) Reported

none

LONG EZSendDigits(EZLine *aLine*, char **pszDigits*, EZCallType *callType*)

Parameters

aLine – **EZLine** device over which digits are to be sent.

pszDigits – Pointer to a NULL terminated character string containing digits to be dialed.

callType – Function call type, should be set to **EZ_SYNC,** or **EZ_ASYNC**.

Description

The function **EZSendDigits** is called to generate DTMF or Pulse digits over an **EZLine** device that is currently connected in an active call. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_CONNECTED** state.

The parameter *pszDigits* should contain the digits to be generated over the **EZLine** device. These must be valid digits that can be generated by the user modem. Valid digits include 0-9, #, *, ! (for hook-flash), and ',' (comma for short delay).

If **EZSendDigits** is called with *callType* **EZ_SYNC**, the function will not return until it the digits have been successfully sent, or some error has occurred. If **EZSendDigits** is called with *callType* **EZ_ASYNC**, the function will return immediately to indicate whether or not the function was successfully started. The client application will be alerted to the overall success or failure of the function through the user defined **EZ_CALLBACK** function.

By default **EZSendDigits** will generate DTMF digits. This functionality can be controlled through the **EZTapiAPI** system parameter **EZ_DIGITMODE_PARAM**. The **EZSetParameter** function can be used to change the default digit generation type from DTMF to Pulse. See Appendix C for more information on **EZ_PARAMETER** system variables and their default settings.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_HANGUP_ERROR, EZ_CALLSTATE_ERROR, EZ_RESOURCE_ERROR**, and **EZ_EVENTMGMT_ERROR**.

EZ_EVENT(s) Reported

EZ_SENDDIGITBEGIN_EVENT, **EZ_SENDDIGITSTOP_EVENT**, **EZ_SENDDIGITEND_EVENT**, and **EZ_UNEXPECTED_EVENT**

LONG EZCollectDigits(EZLine *aLine*, char **pszDigitBuff*, DWORD *dwNumDigits*, char **pszStopDigits*, EZCallType *callType*)

Parameters

aLine – **EZLine** device on which digit collection is to be performed.

pszDigitBuff – Pointer to a NULL terminated character string large enough to hold the total number of digits to be collected. This parameter can be set to NULL if function is called with callType **EZ_ASYNC**.

dwNumDigits – The total number of digits to collect. Must be greater than zero and less than or equal to **EZ_DIGITBUFFERSIZE**.

pszStopDigits – Pointer to a NULL terminated character string containing digits that will cause digit collection to terminate. This parameter should be set to NULL if stop digits are not used.

callType – Function call type, should be set to **EZ_SYNC,** or **EZ_ASYNC**.

Description

The function **EZCollectDigits** is called to collect digits over an **EZLine** device that is currently connected in an active call. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_CONNECTED** state.

The second parameter *pszDigitBuff* will contain the digits collected by the **EZTapiAPI** engine on function completion.

EZCollectDigits will terminate when the number of digits specified by *dwNumDigits* has been collected, when a digit matching an entry in the character string *pszStopDigits* is recognized, or a timeout occurs. If *pszStopDigits* is set to NULL, then no stop digits are valid for this function call, and the function will not return until the number of specified digits have been collected or a timeout occurs.

A timeout will occur if the no digit is collected within the time period specified by the system variable **EZ_FIRSTDIGITTIMEOUT_PARAM**, or if subsequent digits are not recognized within the time period specified by the system variable **EZ_INTERDIGITTIMEOUT_PARAM**. By default **EZ_FIRSTDIGITTIMEOUT_PARAM** is set to 10000 msec, and **EZ_INTERDIGITTIMEOUT_PARAM** is set to 8000 msec. These values can be modified via the **EZSetParameter** function. See Appendix C for more information on **EZ_PARAMETER** system variables and their default settings.

Each **EZLine** device maintains its own internal digit buffer of size **EZ_DIGITBUFFERSIZE**. It is the responsibility of the client application to clear this digit buffer between calls to **EZCollectDigits**. If the number of digits specified for digit collection already exist in the **EZLine**'s digit buffer then the function will return immediately with those digits previously collected.

If **EZCollectDigits** is called with *callType* **EZ_SYNC**, the function will not return until digit collection is complete, or some error occurs. If **EZCollectDigits** is called with *callType*

EZ_ASYNC, the function will return immediately to indicate whether or not the function was successfully started. The client application will be alerted to the overall success or failure of the function through the user defined **EZ_CALLBACK** function.

By default **EZCollectDigits** will collect DTMF digits. This functionality can be controlled through the **EZTapiAPI** system parameter **EZ_DIGITMODE_PARAM**. The **EZSetParameter** function can be used to change the default digit collection type from DTMF to Pulse. See Appendix C for more information on **EZ_PARAMETER** system variables and their default settings.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, **EZ_HANGUP_ERROR, EZ_CALLSTATE_ERROR, EZ_RESOURCE_ERROR, EZ_EVENTMGMT_ERROR**, **EZ_MEMORY_ERROR**, and **EZ_TIMEOUT_ERROR**.

EZ_EVENT(s) Reported

EZ_COLLECTDIGITBEGIN_EVENT, **EZ_COLLECTDIGITEND_EVENT**, **EZ_COLLECTDIGITSTOP_EVENT**, **EZ_DIGITTIMEOUT_EVENT**, and **EZ_UNEXPECTED_EVENT**

Note: If **EZCollectDigits** terminates due to a timeout, even though DTMF digits were sent to the **EZLine** device, the device may not be in the correct listening mode for digit collection. Calling the function **EZPlayPrompt** or **EZRecordPrompt** before attempting digit collection should correct this problem.

LONG EZFlushDigitBuffer(EZLine *aLine*)

Parameters

aLine – **EZLine** device digit buffer to be cleared.

Description

The function **EZFlushDigitBuffer** is called to clear the digit buffer associated with an **EZLine** device specified by the parameter *aLine*.

The **EZTapiAPI** engine maintains an internal digit buffer for all initialized **EZLine** devices. Recognized DTMF or Pulse digits are appended to this buffer whenever digits are recognized while digit monitoring is active.

Digit monitoring is active during calls to **EZCollectDigits**, and calls to **EZPlayPrompt** and **EZRecordPrompt** with **EZTapiAPI** system variable **EZ_DIGITSTOP_PARAM** set to 1.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**, and **EZ_MEMORY_ERROR**.

EZ_EVENT(s) Reported

none

6.4Utility Functions

LONG EZGetCallState(EZLine *aLine*)

Parameters

aLine – **EZLine** obect being queried for its current **EZ_CALLSTATE**

Description

The function **EZGetCallState** is called to retrieve **EZ_CALLSTATE** information for an **EZLine** device.

The parameter *aLine* is used to specify the **EZLine** device being queried for its current **EZ_CALLSTATE**.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is a valid **EZ_CALLSTATE_** constant. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZGetCallInfo(EZLine *aLine*, EZCallInfo **pCallInfo*)

Parameters

aLine – **EZLine** device being queried for **EZCallInfo** data.

pCallInfo – Pointer to an **EZCallInfo** structure to hold caller id information for an answered call.

Description

The function **EZGetCallInfo** is called to retrieve **EZCallInfo** data for an incoming call. This service must be provisioned on the phone line and supported by the underlying modem hardware.

The parameter *aLine* is used to specify the **EZLine** device on which caller id information is being requested. If this function completes successfully, the parameter *pCallInfo* will point to an **EZCallInfo** structure that will hold the returned caller id information for the last incoming call received on the **EZLine** device.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZTestRecordFormat(EZLine *aLine*, EZWaveFormat *pstWaveFormat)

Parameters

aLine – **EZLine** device being queried for the name of its last recorded wave file.

pstWaveFormat – Pointer to an **EZWaveFormat** structure being tested.

Description

The function **EZTestRecordFormat** is called to test the audio capabilities of the voice modem connected to the **EZLine** device. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_CONNECTED** state.

The parameter *aLine* is used to specify the voice modem via its connected **EZLine** device. The parameter *pstEZWaveFormat* is used to specify the wave file format being checked. See Appendix D of this manual for more information on the **EZWaveFormat** structure.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If the specified **EZWaveFormat** is supported by the voice modem device then the return value is **EZ_SUCCESS**. If the **EZWaveFormat** is not supported the return value is **EZ_BADFORMAT_ERROR**. Possible other return values include: **EZ_PARAM_ERROR**, and **EZ_RESOURCE_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZTestPlayFormat(EZLine *aLine*, EZWaveFormat **pstWaveFormat*)

Parameters

aLine – **EZLine** device being queried for the name of its last recorded wave file.

pstWaveFormat – Pointer to an **EZWaveFormat** structure being tested.

Description

The function **EZTestPlayFormat** is called to test the audio capabilities of the voice modem connected to the **EZLine** device. The **EZLine** device specified by *aLine* must be in the **EZ_CALLSTATE_CONNECTED** state.

The parameter *aLine* is used to specify the voice modem via its connected **EZLine** device. The parameter *pstEZWaveFormat* is used to specify the wave file format being checked. See Appendix D of this manual for more information on the **EZWaveFormat** structure.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If the specified **EZWaveFormat** is supported by the voice modem device then the return value is **EZ_SUCCESS**. If the **EZWaveFormat** is not supported the return value is **EZ_BADFORMAT_ERROR**. Possible other return values include: **EZ_PARAM_ERROR**, and **EZ_RESOURCE_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZGetRecordPromptFileName(EZLine *aLine*, char **pszFileName*)

Parameters

aLine – **EZLine** device being queried for the name of its last recorded wave file.

pszFileName – Pointer to a NULL terminated character string to hold the name of the last recorded wave file.

Description

The function **EZGetRecordPromptFileName** is called to retrieve the file name of the last recorded wave file on an **EZLine** device.

The parameter *aLine* is used to specify the **EZLine** device on which **EZRecordPrompt** has completed. If this function completes successfully, the parameter *pszFileName* will hold the name of the last recorded wave file on the **EZLine** device.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZGetCollectDigitsBuffer(EZLine aLine, char *pszBuffer)

Parameters

aLine – **EZLine** device being queried for its digit buffer.

pszBuffer – Pointer to a NULL terminated character string to hold digits collected.

Description

The function **EZGetCollectDigitsBuffer** is called to retrieve collected digits for an **EZLine** device.

The parameter *aLine* is used to specify the **EZLine** device on which digit monitoring was active. If this function completes successfully, the parameter *pszBuffer* will hold the collected digit buffer for the **EZLine** device.

Calling **EZGetCollectDigitsBuffer** will not clear the digit buffer for an **EZLine**. This is accomplished by calling **EZFlushDigitBuffer**.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**. Otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZGetParameter(EZ_PARAMETER *ezParam*, DWORD **pdwValue*)

Parameters

ezParam – **EZ_PARAMETER** value to be retrieved.

pdwValue – Pointer to **DWORD** that is gets set to the current value of the specified **EZ_PARAMETER**.

Description

The function **EZGetTapiAPIParam** is called to get the current setting for a specified **EZ_PARAMETER**. Supported **EZ_PARAMETER**(s) along with their default system values are listed in Appendix C of this manual.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**, otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**.

EZ_EVENT(s) Reported

none

LONG EZSetParameter(EZ_PARAMETER *ezParam*, DWORD *ezValue*)

Parameters

ezParam – **EZ_PARAMETER** to be set.

ezValue – Value that **EZ_PARAMETER** should be set to.

Description

The function **EZSetTapiAPIParam** is used to change the setting for a specified **EZ_PARAMETER**. Supported **EZ_PARAMETER**(s) along with their default system values are listed in Appendix C of this manual. Changing system variables will change the behavior of the **EZTapiAPI**.

This function runs synchronously and has no asynchronous equivalent.

Return Values

If this function call is successful the return value is **EZ_SUCCESS**, otherwise the return value is negative. Possible return values include: **EZ_PARAM_ERROR**.

EZ_EVENT(s) Reported

none

Appendix A – EZTapiAPI Events

EZ_EVENT	Meaning
EZ_RING_EVENT	Signals that an incoming ring signal was received on an **EZLine** device. The third parameter to the user defined **EZ_CALLBACK** function will hold a pointer to a DWORD containing the ring count for the new call.
EZ_CALLCONNECT_EVENT	Signals that an outbound call on an **EZLine** device has been answered.
EZ_CALLNOANSWER_EVENT	Signals that there was no answer for an outbound call attempt.
EZ_CALLBUSY_EVENT	Signals that a busy signal was received on an outbound call attempt.
EZ_CALLERID_EVENT	Caller Id information has been received on an **EZLine** device. The third parameter to the user defined **EZ_CALLBACK** function will hold a pointer to an **EZCallInfo** structure.
EZ_CALLANSWER_EVENT	An incoming call was successfully answered on an **EZLine** device.
EZ_CALLHANGUP_EVENT	A hangup was recognized on the remote end of a telephone call.
EZ_PLAYBEGIN_EVENT	Signals that a playback of a voice prompt has begun on an **EZLine** device.
EZ_PLAYSTOP_EVENT	Signals that playback of a voice prompt of a prompt has been stopped on an **EZLine** device.
EZ_PLAYEND_EVENT	Signals that playback of a prompt has completed successfully on an **EZLine** device.
EZ_RECORDBEGIN_EVENT	Signals that recording of a voice prompt has begun on an **EZLine** device.
EZ_RECORDSTOP_EVENT	Signals that recording of a voice prompt has been stopped on an **EZLine** device. The third parameter to the user defined **EZ_CALLBACK** function will hold a pointer to a NULL terminated character string containing the name of the record file.
EZ_RECORDEND_EVENT	Signals that recording of a voice prompt has completed successfully on an **EZLine** device. The third parameter to the user defined **EZ_CALLBACK** function will hold a pointer to a NULL terminated character string containing the name of the record file.
EZ_COLLECTDIGITBEGIN_EVENT	Signals that digit collection has begun on an **EZLine** device.

EZ_COLLECTDIGITSTOP_EVENT	Signals that digit collection has been stopped on an **EZLine** device. The third parameter to the user defined **EZ_CALLBACK** function will hold a pointer to a NULL terminated character string containing the collected digits.
EZ_COLLECTDIGITEND_EVENT	Signals that digit collection has completed successfully on an **EZLine** device. The third parameter to the user defined **EZ_CALLBACK** function will hold a pointer to a NULL terminated character string containing the collected digits.
EZ_COLLECTDIGITTIMEOUT_EVENT	Signals that a timeout has occurred on an **EZLine** device during digit collection. The third parameter of the **EZ_CALLBACK** function contains a pointer to a character string holding the collected digits.
EZ_SENDDIGITBEGIN_EVENT	Signals that digit generation has begun on an **EZLine** device.
EZ_SENDDIGITSTOP_EVENT	Signals that digit generation has been stopped on an **EZLine** device.
EZ_SENDDIGITEND_EVENT	Signals that digit generation has completed successfully on an **EZLine** device.
EZ_UNEXPECTED_EVENT	Signals that some unexpected error has occurred on an **EZLine** device.

Appendix B – EZTapiAPI Errors

Return Code/Error	Meaning
EZ_SUCCESS	If **EZCallType** is specified as **EZ_SYNC** this return value indicates that the **EZTapiAPI** function has completed successfully. If **EZCallType** is specified as **EZ_ASYNC** this return value indicates that the **EZTapiAPI** function was successfully started.
EZ_ERROR	Generic error, indicating that a function call did not complete successfully. Used internally by the **EZTapiAPI** engine.
EZ_PARAM_ERROR	A parameter to an **EZTapiAPI** function was specified incorrectly.
EZ_MEMORY_ERROR	An internal **EZTapiAPI** memory failure occurred.
EZ_TSP_ERROR	No Telephony Service Provider or supporting modem hardware found.
EZ_TAPIVERSION_ERROR	Current TAPI version will not work with **EZ_CONSOLE** applications.
EZ_EVENTMGMT_ERROR	An internal **EZTapiAPI** Event failure occurred.
EZ_RESOURCE_ERROR	A function call could not be completed or is not supported on an **EZLine** device.
EZ_TIMEOUT_ERROR	A timeout occurred during **EZCollectDigits**.
EZ_CALLSTATE_ERROR	**EZLine** device was not in the proper call state to complete the **EZTapiAPI** function call.
EZ_CALLNOANSWER_ERROR	**EZMakeCall** failed due to no answer.
EZ_CALLBUSY_ERROR	**EZMakeCall** failed due to line BUSY.
EZ_HANGUP_ERROR	A remote hang-up occurred on the **EZLine** device.
EZ_STOP_ERROR	**EZPlayPrompt** or **EZRecordPrompt** has been stopped on an **EZLine** device.
EZ_FILENOTFOUND_ERROR	The file specified for playback to **EZPlayPrompt** does not exist.
EZ_BADFORMAT_ERROR	If returned from **EZPlayPrompt** this error indicates that the wave file specified for playback is corrupt, or not supported by the underlying hardware. *(User should try converting the sample rate of the wave file. Most modems will accept 8kHz, Mono, 16 (or 8) bit wave files for playback and recording.)* If returned from **EZRecordPrompt** this error indicates that the **EZWaveFormat** is not valid, or that the sound device is unable to record voice data at the sample rate defined by

	EZWaveFormat. *(User should try an alternate sample rate for the wave file. Most modems will accept 8kHz, Mono, 16 (or 8) bit wave files for playback and recording.)*
EZ_BEEP_ERROR	**EZRecordPrompt** failed due to not being able to generate beep signal.

Appendix C – EZTapiAPI System Parameters

EZ_PARAMETER	Default Setting	Meaning
EZ_DIGITMODE_PARAM	**EZ_DTMF**	Controls the operation of sending and collecting digits using the **EZTapiAPI** function calls **EZSendDigits** and **EZCollectDigits**. Allowed Values: **EZ_DTMF**, **EZ_DTMFEND**, **EZ_PULSE**
EZ_DIGITWAVESTOP_PARAM	1	Controls whether or not prompt playback and recording will stop on recognition of a DTMF or Pulse digit. Change to zero to disable this feature.
EZ_FIRSTDIGITTIMEOUT_PARAM	10000	The time in which the first digit must be recognized during a call to **EZCollectDigits** before a digit collection timeout occurs. This value is in milli-seconds.
EZ_INTERDIGITTIMEOUT_PARAM	8000	The time in which subsequent digits must be recognized during a call to **EZCollectDigits** before a digit collection timeout occurs. This value is in milli-seconds.
EZ_RECORDHANGUPTRIM_PARAM	0	The amount of time to remove from the end of a voice recording that ends due to recognition of dial-tone after a remote hangup. This value is in milli-seconds.
EZ_RECORDDIGITTRIM_PARAM	0	The amount of time to remove from the end of a voice recording that ends due to recognition of a DTMF or Pulse digit when **EZ_DIGITWAVESTOP_PARAM** is set to 1. This value is in milli-seconds.

Appendix D – EZTapiAPI Definitions and Data Types

The following information is taken directly from the header files ezerrors.h, and ezparams.h. These header files contain the **EZTapiAPI** definitions, and data types.

```
#ifndef _EZERRORS_H_
#define _EZERRORS_H_


// return codes
#define EZ_SUCCESS                          0L
#define EZ_ERROR                           -1L

// EZTapiAPI Errors;
#define EZ_PARAM_ERROR                     -2L
#define EZ_MEMORY_ERROR                    -3L
#define EZ_TSP_ERROR                       -4L
#define EZ_TAPIVERSION_ERROR               -5L
#define EZ_EVENTMGMT_ERROR                 -6L
#define EZ_RESOURCE_ERROR                  -7L
#define EZ_TIMEOUT_ERROR                   -8L
#define EZ_CALLSTATE_ERROR                 -9L
#define EZ_CALLNOANSWER_ERROR             -10L
#define EZ_CALLBUSY_ERROR                 -11L
#define EZ_HANGUP_ERROR                   -12L
#define EZ_STOP_ERROR                     -13L
#define EZ_FILENOTFOUND_ERROR             -14L
#define EZ_BADFORMAT_ERROR                -15L
#define EZ_BEEP_ERROR                     -16L


#endif // _EZERRORS_H_
```

```
#ifndef _EZPARAMS_
#define _EZPARAMS_


// Modem Type
typedef enum
{
    EZ_MODEMTYPE_UNKNOWN,
    EZ_MODEMTYPE_DATA,
    EZ_MODEMTYPE_VOICE
} EZModemType;

// Client Application Type
typedef enum
{
    EZ_CONSOLE_APP,
    EZ_WINDOWS_APP
} EZAppType;


typedef enum
{
    EZ_MONITOR,
    EZ_OWNER
} EZCallPrivilege;

// Mode of Operation Values
typedef enum
{
    EZ_SYNC,
    EZ_ASYNC
} EZCallType;


typedef enum
{
    EZ_DTMF,
    EZ_DTMFEND,
    EZ_PULSE
} EZDigitMode;


// EZTapiAPI engine parameters
typedef enum
{
    EZ_DIGITMODE_PARAM,
    EZ_DIGITWAVESTOP_PARAM,
    EZ_FIRSTDIGITTIMEOUT_PARAM,
    EZ_INTERDIGITTIMEOUT_PARAM,
    EZ_RECORDHANGUPTRIM_PARAM,
    EZ_RECORDDIGITTRIM_PARAM
} EZ_PARAMETER;


// EZTapiAPI Call States
typedef enum
{
    EZ_CALLSTATE_UNKNOWN,
    EZ_CALLSTATE_ONHOOK,
    EZ_CALLSTATE_WAITFORCALL,
    EZ_CALLSTATE_ANSWERCALL,
    EZ_CALLSTATE_MAKECALL,
    EZ_CALLSTATE_CONNECTED,
    EZ_CALLSTATE_HANGUPCALL,
    EZ_CALLSTATE_PLAYMESSAGE,
    EZ_CALLSTATE_RECORDMESSAGE,
    EZ_CALLSTATE_COLLECTDIGITS,
    EZ_CALLSTATE_SENDDIGITS
} EZCallState;
```

```
// EZLine identifies different lines using EZTapiAPI
typedef DWORD EZLine;

// EZTapiAPI Events
typedef DWORD EZ_EVENT;

// EZTapiAPI defined message to be posted back to VB Clients
#define EZ_EVENT_MSG                    0x00000401

#define EZ_RING_EVENT                   0x00000001
#define EZ_CALLCONNECT_EVENT            0x00000002
#define EZ_CALLNOANSWER_EVENT           0x00000004
#define EZ_CALLBUSY_EVENT               0x00000008
#define EZ_CALLERID_EVENT               0x00000010
#define EZ_CALLANSWER_EVENT             0x00000020
#define EZ_CALLHANGUP_EVENT             0x00000040
#define EZ_PLAYBEGIN_EVENT              0x00000080
#define EZ_PLAYSTOP_EVENT               0x00000100
#define EZ_PLAYEND_EVENT                0x00000200
#define EZ_RECORDBEGIN_EVENT            0x00000400
#define EZ_RECORDSTOP_EVENT             0x00000800
#define EZ_RECORDEND_EVENT              0x00001000
#define EZ_COLLECTDIGITBEGIN_EVENT      0x00002000
#define EZ_COLLECTDIGITSTOP_EVENT       0x00004000
#define EZ_COLLECTDIGITEND_EVENT        0x00008000
#define EZ_COLLECTDIGITTIMEOUT_EVENT    0x00010000
#define EZ_SENDDIGITBEGIN_EVENT         0x00020000
#define EZ_SENDDIGITSTOP_EVENT          0x00040000
#define EZ_SENDDIGITEND_EVENT           0x00080000
#define EZ_UNEXPECTED_EVENT             0x00100000


typedef struct
{
    int NumChannels;
    int SampleRate;
    int BitsPerSample;
} EZWaveFormat;


// Miscellaneous size values for array declaration
#define EZ_MAXPATHSIZE                  80
#define EZ_DIGITBUFFERSIZE              50


#define EZ_CALLINFOSIZE                 30

typedef struct
{
    char CallerName[EZ_CALLINFOSIZE];
    char CallerNumber[EZ_CALLINFOSIZE];
} EZCallInfo;



#endif // _EZPARAMS_
```

Appendix E – Demo Applications

Included with the **EZTapiAPI SDK** are several demo applications designed to demonstrate the **EZTapiAPI** function calls. Developers are encouraged to build and run these sample applications to get them started building their own Interactive Voice Response (IVR) Systems, and other Telephony related applications with the **EZTapiAPI SDK**.

Example Programs when installed to their default location can be found at:

"\Program Files\PC Phone Connections\EZTapiAPI SDK\Samples"

Samples include:

"..\Samples\Console"	*This the console demo application discussed in the Programming Model section of this manual.*
"..\Samples\MFC"	*MFC Windows Demo Application.*
"..\Samples\VB"	*VB Windows Demo Application.*

Before running any of these applications the user may need to edit their *Path* system environment variable to include the location of the **EZTapiAPI** Library. The default location for this library is "\Program Files\ PC Phone Connections\EZTapiAPI SDK\Lib".

All wave file recordings must co-exist in the same directory as the demo executable that uses it. One special wave file *beep.wav* is used by the EZTapiAPI function **EZRecordPrompt**. Since Unimodem does not support the TAPI function **lineGenerateTone**, this file is included to play the beep tone before message recording. You will need this recording if you want to play a beep notification to callers before recording voice messages in your own applications.

Appendix F – Debug Mode

The **EZTapiAPI** Library can be run in two modes, normal operating mode, and special debug mode. Enabling the special debug mode is strongly discouraged. In this operating mode the **EZTapiAPI** library performs a large number of disk I/O operations, that are un-necessary during normal operation and could affect system performance.

If asked to enable this mode by a support person from PC Phone Connections, you will need to define the following system environment variables: *eztapilog*, and *eztapiloglevel*. See the documentation that came with your computer operating system for specific instructions on enabling these variables. The environment variable *eztapilog* should be set to True, i.e. eztapilog=true. The environment variable *eztapiloglevel* should be set to one of 3 values: "error", "warning", or "info", i.e. *eztapiloglevel*=error, *eztapiloglevel*=warning, or *eztapiloglevel*=info.

Setting these variables will cause the **EZTapiAPI** to generate a log file named *EZTapiAPI.log*. The log file will be created in the directory from which an **EZTapiAPI** compiled application is run.