

TinyTERM Plus

version 4.42

Data Exchange Between Windows Programs

CScript Programmers Guide

TE Control Object Reference

FT Control Object Function Reference

TERM Script to CScript Translation Guide

Copyright Notice

Copyright 2005 by Century Software, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the written prior permission of Century Software, 5284 South Commerce Drive, Suite C-134, Salt Lake City, Utah 84107.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013.

Trademarks

Century Software TinyTERM, TinyTERM PLUS and the "hub" logo are trademarks or registered trademarks of Century Software, Inc. MS-DOS, MS Windows, and XENIX are trademarks of Microsoft, Inc. UNIX is a registered trademark of AT&T., DEC is a registered trademark of Digital Equipment Corporation. Ethernet is a trademark of Xerox Corporation. VT320, VT220, VT100, VT52, VAX and ALL-IN-1 are trademarks of Digital Equipment Corporation. Compuserve and Compuserve B are trademarks of Compuserve Incorporated. IBM is a registered trademark of International Business Machines. Sequent is a trademark of Sequent Computers, Inc. Wyse 50 and Wyse 60 are trademarks of Wyse Technology, Inc. All other trademarks, trade names, or company names referenced herein are used for identification purposes only and are the property of their respective owners.

Disclaimer

Century Software makes no representations or warranties with respect to this product or the contents hereof. This product (including both software and this manual) is sold as is and without any express warranty of any nature. ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS HEREBY DISCLAIMED.

Limitation of Liability

In no event shall Century Software be liable for consequential damages, even if Century Software has been advised of the possibility of such damages. Further, Century Software reserves the right to revise this publication and to make changes from time to time in the contents hereof without obligation of Century Software to notify any person or organization of such revision or changes.



CENTURY

S O F T W A R E

5284 South Commerce Dr. Suite C-134
Salt Lake City, Utah 84107

SECURE CONNECTIVITY SOLUTIONS

Toll Free (800) 877-3088
Phone (801) 268-3088
Fax (801) 268-2772
E-mail sales@centurysoftware.com
Web www.centurysoftware.com

Copyright © 1985-2005 Century Software, Inc. All Rights Reserved.
Century Software, TinyTERM Plus and the "hub" logo are trademarks or registered trademarks of Century Software, Inc. All other trademarks, trade names, or company names referenced herein are used for identification purposes only and are the property of their respective owners.

Table of Contents

Introduction

TinyTERM Plus Scripting Languages	i
Changing TinyTERM Plus's User Interface.....	i
TinyTERM Plus Automation.....	ii
TinyTERM Plus in Other Programs.....	ii
Chapters in this Guide.....	ii

Chapter 1

Data Exchange Between Windows Programs

The Clipboard.....	1
Dynamic Data Exchange	2
Server Commands	3
System Topic.....	3
Script Topic.....	4
Terminal Topic	4

Chapter 2

CScript Programmers Guide

Using the CScript Scripting Language.....	5
Script Overview.....	5
JavaScript Compatible	5
Creating and Executing Scripts	5
Examples and Samples.....	7
Referenced Publications.....	7
CScript Builtin Function Reference	8
Overview.....	8
Conventions	8
Value Abbreviations.....	8
CScript Command and Operators.....	9
Primitive Commands	10
File I/O Commands.....	11
User Interface Commands.....	19
Utility Commands	21
Win32 Commands.....	27
TinyTERM Commands\.....	32
Javascript Commands.....	37
C Commands.....	39
DDE Commands	39

Chapter 3**TE Control Object Reference**

Properties	42
Function Reference	49
Events Reference	60

Chapter 4**FT Control Object Function Reference**

File Transfer Session Management	63
Properties	63
Get Properties	63
Set Properties	64
Function Reference	66
File Transfer Initiation Functions	66
File Transfer Utility Functions	68
Events Reference	69
File Transfer Callback Functions	69
Sample File Transfer Script	71

Chapter 5**TERM Script to CScript Translation Guide**

Translating TERM Script Language to CScript	72
Running TERM Scripts with TinyTERM	72
Executing TERM Script Language	72
Translation Errors	73
Unsupported TERM Script Language Features	73
Translation Tips	73
Command Line TSL Script Translator	78
Installation	78
Usage	78
Known Problems	79
Uninstalling	79
TSL/CSL Translation Reference	80
Configuration Settings	80
Current Connection Configuration Settings	85
User Interface Commands	86
File Transfer Commands	88
General Commands	89
Operating System Level Commands	91
File I/O Commands	91
Miscellaneous Commands	92
Control and Flow Commands	92
Previous Version Retrofitted Commands	93
TERM Script Language Functions	94

Introduction

In this Introduction

TinyTERM Plus Scripting Languages

Changing TinyTERM Plus's User Interface

TinyTERM Plus Automation

TinyTERM Plus in Other Programs

Chapters in this Guide

Welcome to the TinyTERM Plus Programmers Reference Guide. Contained within this guide, you will find details on how to take advantage of TinyTERM Plus's ability to run small programs, called scripts, to enhance the functionality of the software. This includes the ability to reconfigure the user interface to your liking, remotely control applications normally controlled through the terminal, and even how to embed portions of TinyTERM Plus in your own software through the use of ActiveX (COM) controls.

TinyTERM Plus Scripting Languages

TinyTERM Plus actually has support for two separate scripting languages. The more modern of the two was introduced with TinyTERM Plus version 4.0, and is called CScript (or CSL.) This language is very similar to, and is very compatible with, JavaScript. If you have experience with JavaScript, learning CScript will be very straightforward.

The older of the two languages is called TERM Script (or TSL.) It was originally introduced with Century's TERM line of products many years ago. It has a much simpler syntax, like that of old BASIC, but may not be as familiar looking to modern users.

Either of the two languages may be used to extend TinyTERM Plus functionality, though CSL is preferred, as it's the language used internally for running TinyTERM Plus. In both languages, commands are case-insensitive.

Changing TinyTERM Plus User Interface

Using CScript, you can change the way that the user interface of TinyTERM Plus behaves. Since the entire TinyTERM Plus product is put together using CScript, modifications to the menus and toolbars are quick and easy, as are adding your own dialogs settings.

Due to the built-in hooks in TinyTERM Plus, scripts may be triggered automatically when certain actions occur. For instance, a confirmation dialog could be added to ensure that a user doesn't disconnect from a host system before logging off, being triggered when a "disconnect" command is issued.

In addition, CScript may be used to lock down the user interface to prevent changes being made to settings, or to restrict the options available for change to the user.

TinyTERM Plus Automation

Complete control over TinyTERM Plus is possible when scripting is used. One of the included sample scripts causes TinyTERM Plus to act like a miniature old-style BBS system, waiting for incoming connections and then presenting menus to the user for manipulating files and leaving messages, all without intervention from the local user.

This same capability can be utilized to automate nightly file transfers, or perform other routine tasks on host systems where this is normally not possible.

TinyTERM Plus in Other Programs

As mentioned above, you can use portions of the TinyTERM Plus product from within your own software. This includes Visual Basic programs and nearly any environment where ActiveX controls can be used. The main functional components of TinyTERM Plus – File Transfer and Terminal Emulation – are self-contained ActiveX controls which may be freely used elsewhere without the complete TinyTERM Plus user interface.

Chapters in this Guide

This guide is divided into three main chapters. The first, Data Exchange Between Windows Programs, delves into the ability to exchange data between TinyTERM Plus and other programs. This allows you to, for instance, connect to a UNIX host and run an application, select data from the application, and return it your spreadsheet.

The second chapter, CScript Programmers Guide, covers all details of programming using CScript. This includes how to edit and run scripts, a syntax guide, a complete listing of all functions available to the user, and reference listings for the ActiveX controls and their script interfaces.

The third chapter, TERM Script to CScript Translation Guide, goes into detail about how TERM Script is handled within TinyTERM Plus, a complete listing of all the TSL functions, and known issues and workarounds for when you are translating TERM Script to CScript.

Chapter 1

Data Exchange Between Windows Programs

In this chapter

- The Clipboard
- Dynamic Data Exchange
- Server Commands
- System Topic
- Script Topic
- Terminal Topic

Windows introduced the ability of performing several tasks almost simultaneously on a single PC. This means that several programs can be accessible at once. With this ability also came two methods of sharing information between programs. The next two sections describe how TinyTERM Plus can share data with other Windows programs.

The Clipboard

Windows provides a generic location for data exchange called the clipboard. The user may use the clipboard to copy data from one application to another. This form of data exchange depends on full user control.

TinyTERM Plus uses the clipboard through the following commands, which are also a part of the main menu: SCREEN COPY, PASTE LINK, PASTE TEXT.

SCREEN COPY (Copy on the menu)

Exports terminal screen text to the clipboard, and link format data that describes the area of the screen to potential Dynamic Data Exchange (DDE) client applications. The next section describes what a DDE client application does. A DDE link may be started by the user, and it may be maintained by the application.

PASTE LINK

Imports link format data from the clipboard and establishes a DDE client conversation. The data of the conversation is copied to the active communications link. This PASTE command utilizes DDE.

PASTE TEXT (Paste on the menu)

Imports text format data from the clipboard and copies it to the active communications port.

Dynamic Data Exchange

Windows provides a second method for separate applications to share data. Dynamic Data Exchange (DDE) is a “conversation” between applications.

The conversation must include a topic, and may include an item (topic and item are described under Server Commands). One application is the DDE client, it requests data and services from the server. The other application is the DDE server, it responds to requests from the client.

TinyTERM Plus supports both client and server activities. TinyTERM Plus may be a DDE client, requesting services or data from a DDE server through the DDE script commands. TinyTERM Plus may also be a DDE server, providing several services to a DDE client.

TinyTERM Plus’s script language DDE client requests include: DDE ADVISE, DDE CLOSE, DDE EXECUTE, DDE INIT, DDE POKE, DDE REQUEST, DDE TIMEOUT, DDE UNADVISE.

A typical DDE conversation from the client side would follow these steps:

1. DDE INIT
2. DDE ADVISE, DDE EXECUTE, DDE REQUEST, DDE POKE or DDE UNADVISE
3. Repeat step 3 for the duration of the DDE conversation, with the exception of DDE ADVISE which creates a dynamic link maintained by the server with no further calls to DDE ADVISE.
4. DDE CLOSE

NOTE: A PASTE LINK encapsulates the above steps in one command. However, PASTE LINK requires that the DDE server place a valid link format entry into the clipboard prior to execution. The function `ispaste()` should be used to determine if a link format entry is in the clipboard before executing the PASTE LINK command.

DDE INIT

Initiates a DDE conversation with a server.

DDE ADVISE, DDE UNADVISE

Requests to be advised when an item changes in the DDE server. An item is a variable owned by the DDE server. The advise notification remains in effect until DDE UNADVISE or DDE CLOSE. This type of link continues to update on an item without further user action or script commands. It’s like an automatic copy and paste between two applications.

DDE EXECUTE

Requests a service from the DDE server. This capability has varying functionality, depending on the DDE server. Consult the documentation of the server application for the services provided.

DDE POKE

Sends data to the DDE server. This has the effect of changing the value in the target item. An item is a variable owned by the DDE server. If the value must be changed many times, there must be a script command for each change.

DDE REQUEST

Requests data from the DDE server. The value contained in the item is returned to the client. This has the effect of copying data from the server to the client. If the data is expected to change, the client must “poll” the server with a script command each time the client needs a “fresh” value. It is better to use DDE ADVISE to monitor a value.

Server Commands

The DDE server responds to requests made by client application. The following paragraphs describe server DDE activities, and the topics and items TinyTERM Plus provides to the DDE client.

DDE ENABLE | DISABLE

Sets whether or not TinyTERM Plus will accept DDE client requests. When enabled TinyTERM Plus will accept client requests on the provided topics. When disabled, TinyTERM Plus ignores all DDE requests.

DDE NAME

Changes TinyTERM Plus’s DDE server name. TinyTERM Plus’s default DDE server name is WTERM. However, if this name conflicts with another DDE server, use this command to change it.

System Topic

TinyTERM Plus’s System topic provides general interest information to a DDE client. Items in this DDE context include: Topics, SysItems, Formats and Help.

Topics

Lists all available topics on which to create a DDE conversation. System is described here. Script and Terminal are described later.

SysItems

Lists the items available for DDE REQUESTs in a conversation based on the System topic. These include: Topics, SysItems, Formats and Help.

Formats

Lists the types of data TinyTERM Plus supports for DDE exchange. TinyTERM Plus supports TEXT and LINK data formats.

Script Topic

TERM's Script topic provides information on the script language. Item in this DDE context include any string. The string is parsed as though read from a TERM script file or from the "Execute Script" menu option. It is possible to compose a TERM script by sending a series of these commands from a DDE client application.

DDE REQUEST topic strings are any valid TERM script expression such as the names of script variables or functions with full parameter lists. Any valid TERM script expression that can be on the right of a SETVAR command is valid.

DDE ADVISE | UNADVISE topic strings may be the name of single script variable. Anything that can be on the left side of a SETVAR command is valid.

DDE POKE topic strings must be the name of a single script variable. However, TERM strings may not exceed 255 characters in length, and may not contain a NUL character.

Terminal Topic

TERM's Terminal topic provides information on emulation window. Items in this DDE context include any string of the format: *XnnnYnnnWnnnHnnn* where *n* is any digit. The string describes an area of the active emulation window based on character units.

DDE REQUEST

Reads the text at the given window location.

DDE ADVISE | UNADVISE

Requests update when the given window area changes.

DDE POKE

Sends the data to the active communications line.

All TERM's topics support the DDE EXECUTE function. The functions provided include the entire script language. This allows complete and total control of TERM by another application.

Chapter 2

CScript Programmers Guide

In this chapter

Using the CScript Scripting Language

CScript Builtin Function Reference

CScript Object Reference: TE Control

CScript Object Function Reference: FT Control

Using the CScript Scripting Language

SCRIPT OVERVIEW

CScript is Century Software's script programming language that is used to manipulate, customize, and automate TinyTERM to extend its functionality. TinyTERM provides a JavaScript-compatible host environment for client-side computation including, for instance, objects that represent terminal emulation and file transfer, along with commands that allow for automated startup, login, logout, file transfers and remote system access.

In that CScript is similar in functionality and syntax to JavaScript, learning CScript is very easy and takes advantage of the vast resource of JavaScript in both print and on the World Wide Web. Included in this document is a list of resources that can be used in conjunction with the CScript reference guide to help you master CScript for custom Web-to-Host access.

JAVASCRIPT COMPATIBLE

Many programming languages such as C++, Java, JavaScript and CScript use objects to organize and address information. These languages are described as object-oriented.

The concept of an object is merely a thing, something that you can see. Using a computer for example, it has many attributes such as brand name, model, processor type, and processor speed. The computer also has uses called properties and methods such as boot up, running an application, or performing a mathematical calculation. JavaScript and CScript think along these same lines.

DOT SYNTAX

JavaScript as well as CScript use dot syntax to access properties and methods of objects. Using the computer as an example one might use the following to describe running an application.

```
Objcomputer.runapp("Excel.Application");
```

CREATING AND EXECUTING SCRIPTS

CScripts are executed from within the TinyTERM emulator by pulling down the Tools menu and selecting Script Editor. When executed, the dialog box shown will be displayed with the following commands.

New

This command is used to load a new, “(Untitled)” .cs file. This command will prompt to save if there is any information in the dialog editor, then clear the dialog editor and create a blank script.

Open

This command is used to open a previously created .cs file. By executing this command, a standard Windows file open dialog is launched that allows viewing of .cs files.

Edit

This command opens the currently selected script in Windows Notepad. The text window (see below) has a 10k limit on script size. The Edit button is provided for larger scripts, and as an alternative editor. It uses the editor specified under Edit - Preferences - User Interface Default Editor. If no editor is specified, it uses Notepad.

Save

Used to save the currently loaded script file. If the currently loaded script is “(Untitled)”, then a name must first be assigned to the script, and so the Save functions as a Save As (see below).

Save As

Used to save an “(Untitled)” script or to save the currently loaded script with a new name. This can be used to make a copy of the currently loaded script.

Run

This command executes the currently loaded script. It will prompt to save before running if the current script has been changed, but not yet saved.

Exit

This command closes the “Execute Script” dialog. It will prompt to save before exiting if the current script has been changed, but not yet saved.

Text window

This dialog box is used to create and edit CScript script files. It can also be used to execute a single script command or a group of commands without saving to a file.

Progress window

This window shows the progress and/or status of recent commands.

You can execute scripts from within the Script Editor dialog box or with the Execute Script File command on the Tools menu.

1. Choose Script Editor from the Tools menu.
 2. Click Open, and choose the script you want to execute.
 3. Click Run.
- or*
1. Click the Execute Script button on the ribbon bar, or choose Execute Script File from the Tools menu.
 2. From the dialog box, choose the script you want to execute and click OK.

You can also set session properties in the TinyTERM emulator’s user interface to execute scripts post-application startup, post-session start, post-connect, post-login, pre-logout, pre-disconnect, pre-session close, and pre-application exit.

As well, you can launch script commands from the host TinyTERM is connected to by preceding the commands with `\E&oF` and following them with `^M`, where `\E` is the escape character (ASCII 27) and `^M` is the Enter character (ASCII 13). For example, if the hosts sends or displays the following:

```
\E&oFte.cls();te.displaynl("OK");^M
```

the TinyTERM screen will be cleared, and the letters “OK” will display in the upper left corner.

ACTIVATION ON COMMAND

TinyTERM allows for automatic script execution during various states of the application.

Application Start

Application Exit

Session Start

Session Exit

Post Login

Pre Logout

Connect

Disconnect

Post Server Start

Pre Server Stop

EXAMPLES AND SAMPLES

There are several example scripts distributed with TinyTERM. These are found in the TinyTERM directory. All have the `.cs` extension.

REFERENCED PUBLICATIONS

JavaScript Bible

Danny Goodman / Paperback / Date Published: January 1998

Synopsis: With all the different JavaScript titles available today, it's pretty easy to get overwhelmed when it comes to finding the right book for you. Wouldn't it be great if there was a complete, comprehensive JavaScript guide for users of all levels? Well, there is: JavaScript™ Bible, 3rd Edition, written by leading JavaScript authority Danny Goodman. www.idgbooks.com

JavaScript: The Definitive Guide

David Flanagan, Dan Shafer / Paperback / Date Published: January 1998

Synopsis: The third edition of this definitive reference covers the latest version of JavaScript—JavaScript 1.2—as supported by Netscape Navigator 4.0. It can be used to help users create dynamic, interactive, Web-based applications that are powered by JavaScript.

CScript Builtin Function Reference

OVERVIEW

The CScript scripting language provides a library of functions that enable you to interact with the operating system and includes utility functions to make developing scripts for the TinyTERM emulator easier. This document describes the commands available and how to use these built in functions.

CONVENTIONS

The CScript Builtin Function Reference describes commands using the following format:

Return _ value **Command**

syntax

Description

Example

VALUE ABBREVIATIONS

Values passed to script functions are categorized by the first letter of their names. The category letters are all lower-case, and follow the convention listed below. The category names also apply to the values returned by the functions.

Category Name	Implementation	Code
-----	-----	-----
Time	32-bit integer	t
Integer	32-bit	i
String	64k max length	s
Floating point	IEEE 64-bit	f
Boolean	0/non-zero True/false	b
Handle (Win32)	32-bit	h
Object	32-bit	o
Void	No parameter or return value	
Variable Name	String	v

CSCRIPT COMMAND AND OPERATORS

CScript supports the following list of JavaScript commands and keywords:

```
break
continue
dimint
dimstr
else
for
function
if
return
var
while
```

CScript supports these additional commands that use syntax borrowed from the C language.

```
case
default
goto
switch
```

All the above commands are documented in the CScript Builtin Function Reference. CScript supports the following operators in the following order from lowest to highest precedence:

```
=, +=, -=, *=, /=, %=, &=,
|=, ^=, <<=, >>=, >>>=    assignment
? :                          conditional
||                            logical or
&&                          logical and
|                            boolean or
^                            boolean xor
&                            boolean and
==, !=                       logical compare
<, <=, >, >=               logical compare
<<                          logical shift left
```

continued...

continued from previous page...

>>	arithmetic (signed) shift right
>>>	logical (zero-filled) shift right
+, -	addition, subtraction
*, /, %	multiplication, division, modulo
-	unary minus
~	boolean not
!	logical not
typeof	return string expression type
void	eval expression and return nil
new	create new object
++	pre/post increment
--	pre/post decrement
[]	property reference, vector index
.	property reference

PRIMITIVE COMMANDS

Boolean **AxRegisterEvent**(*oObject, str, str2*)

Registers call back function.

```
AxRegisterEvent(ft,"EndBatchReceive","ft_EndBatchReceive");
```

Object **CreateObject**(*sObjName, sObjClass, iExStyle, iStyle, iX, iY, iW, iH, sTitle, iID, oParentObject*)

Creates an object based on a Win32 window class.

```
ObjFrame = CreateObject( "TE_Frame", "[FRAME]", 0, WS_OVERLAPPEDWINDOW, 0, 0, 0, 0,  
"TinyTERM", 0, 0 );
```

Void **DestroyObject**(*oObj*)

Deletes and deallocates an object.

```
DestroyObject(ft);
```

Integer **RegisterFile**(*sFile*)

Registers an in-process COM server in DLL sFile. Returns 0 for success or -1 for failure.

```
success = RegisterFile("cencom32.dll");
```

Void **Dprintln**(*sArg0*[[*sArg1*],*sArg2*],*sArg3*...)

Prints to debug screen in debug mode.

```
dprintln("Hello");
```

FILE I/O COMMANDS

Boolean **Chdir**(*sPath*)

Changes the current directory.

```
chdir("C:\temp");
```

Integer **Copy**(*sSrcFile*, *sDestFile*)

Copies one file to another. Returns 0 on success.

```
copy("default.tpx", "newtpx.tpx");
```

Integer **Erase**(*sFile*)

Erases file. Returns 0 on success.

```
erase("default.tpx");
```

Boolean **Exists**(*sFileName*)

Returns true if passed file exists.

```
b = exists("d:\century\default.tpx");
```

Void **Fclose**(*iFileID*)

Closes a file opened with Fcreate or Fopen.

```
fclose(1);
```

String **Fconcat**(*sPath*,*sFile*,*sExt*)

Returns a full pathname from path, file and extension. Note: this function will automatically concatenate a "\ " character to the end of *sPath* unless *sPath* ends with a "\ " or contains a drive designation, such as "c:".

```
str = fconcat("c:\century","default","tpx");
```

Boolean **Fcreate**(*iFileID, sFile*)

Creates a new file. Overwrites an existing file.

```
fcreate(iFileID, "default.tpx");
```

Boolean **Feof**(*iFileID*)

Returns true if at end of file.

```
b = feof(fileno);
```

String **Fext**(*sPathFile*)

Returns extension from passed file string.

```
str = fext("c:\\century\\default.tpx");
```

Integer **Fflush**(*iFilenum*)

Flushes the data for the file associated with iFilenum. If the file is open for output, this will write the contents of the buffer for the file to disk. If the stream is open for input, this will clear the contents of the buffer. Returns 0 for completion, 1306 if the file number is not in the range of 1 through 20, and 1303 if an I/O error occurred during the flush operation.

```
str = fext("c:\\century\\default.tpx");
```

String **Fhead**(*sNamePath*)

Gets file head.

```
fileh = fhead("d:\\century\\default.tpx");
```

Time Value **FileDate**(*sFileName*)

Gets file date of filename as a time value.

```
tim = filedate("default.tpx");
```

Integer **FileMode**(*sFileName*)

Returns the mode settings for file *sFileName*. This is a 16-bit value returned from a `stat` call to the file. Possible values include:

Regular file	0x8000 (32768)
Directory	0x4000 (16384)
Character special	0x2000 (8192)
Pipe	0x1000 (4096)
Read permission	0x0100 (256)
Write permission	0x0080 (128)
Execute permission	0x0040 (64)

The returned value will be a sum of one or more of these settings.

```
mode = FileMode("default.tpx");
```

Integer **FileSize**(*sFileName*)

Gets the size in bytes of a file. If the file does not exist, a 0 will be returned.

```
bytes = FileSize("default.tpx");
```

Integer **Fopen**(*iFileID*, *sFile*, *sMode*)

Opens a file *sFile*, sets the file ID for that file to *iFileID*, and opens the file read/write mode according to *sMode*.

iMode

RA	Read only, ASCII mode
RB	Read only, Binary mode
WA	Write to new file, ASCII mode (<i>overwrites an existing file</i>)
WB	Write to new file, Binary mode (<i>overwrites an existing file</i>)
UA	Append to an existing file, ASCII mode
UB	Append to an existing file, Binary mode

Note that opening a text file with Unix-style line breaks in ASCII mode can lead to some unexpected behavior, including `ftell()` returning negative values and `fseek()` not moving the file pointer predictably. Use Binary mode for files with Unix-style line breaks.

```
fopen( 19, "default.tpx", "RB" );
```

String **Fname**(*sNamePath*)

Returns stripped filename only from passed full pathname.

```
str = fname("d:\\century\\default.tpx");
```

String **Fpath**(*sNamePath*)

Returns stripped path only from passed full path and filename.

```
path = fpath("d:\\century\\default.tpx");
```

String **Fread**(*iFileID*, *iByte*)

Reads *iByte* bytes from file referred to by *iFileID*. If *iByte* = -1, reads to end of line or file, whichever comes first.

```
str = fread( 1, -1 );
```

String **Freadln**(*iFileID*, *iByte*)

Reads a line from a file referred to by *iFileID*, either to end of line or a maximum of *iByte* bytes, whichever comes first.

```
str = freadln( 1, -1 );
```

Integer **Fseek**

Fseek(*iFileID*, *iOffset*, *iOrigin*)

Moves file pointer for file *iFileID* to location *iOffset* from the location set by *iOrigin*.

iOrigin

0 Beginning of file

1 Current position

2 End of file

```
fseek(1,3,0);
```

Integer **Ftell**(*iFileID*)

Gets the current position of the file pointer in the file referred to by *iFileID*.

```
nPosition = ftell( 2 );
```

Integer **Fwrite**(*iFileID*, *sText*, *iByte*)

Writes *sText* to file associated with *iFileID*, with *iBytes* maximum number of bytes written. If *iBytes* is negative, **Fwrite** writes the all of *sText*. Returns 0 for success, 1 if the file is not open, and 1300 for an I/O error writing to the file.

```
fwrite( 2, "Hello", -1 );
```

Integer **Fwriteln**(*iFileID*, *sText*, *iByte*)

As **Fwrite**, but adds a carriage return/line feed pair to the end *sText* when writing.

```
fwriteln( 2, "Hello", -1 );
```

String **GetCommandLine**()

Gets the command line, including all parameters, used to launch this session of the TinyTERM emulator.

```
cmdline = GetCommandLine( );
```

String **GetPath**(*sFile*,*sType*)

Gets the path of a file. *sType* can be "U" (user), "S" (system) or both. The first listed has precedence.

```
path = GetPath("system.ini", "S");
```

Void **IniDeleteSection**(*sFile*, *sSection*)

Deletes a section from an .ini file.

```
IniDeleteSection( "user.ini", "keyboard" );
```

Void **IniDelStr**(*sSection*,*sKeyName*)

Deletes a key from an ini-format file.

```
IniDelStr("colors","fgcolor");
```

String **IniEnumSections**(*sFile*,*sMatch*)

Returns a "]" delimited list of all sections in the .ini file that match the pattern. (Note that the returned strings in the list have the matched suffixes removed.)

```
list = inienumsections( "log.dat", "*.login" );
```

String **IniGetStr**(*sSection,sKeyName,sDefault*)

Gets data from an ini-format file. Returns default if key does not exist.

```
str = inigetstr("log",key, " ");
```

String **IniListKeys**(*sFile,sSection*)

List keys defined in an ini-format file.

```
list = IniListKeys("tt.ini","section");
```

Void **IniMerge**(*sSect,sFile,sSect2,sFile2*)

Merges sSect from file sFile into sSect2 in sFile2.

```
inimerge("login","tt.ini","login","ftp.ini");
```

Void **IniReplaceKeys**(*sFile,sSection,sList*)

Replaces keys in an ini-format file.

```
IniReplaceKeys("tt.ini",section,keylist);
```

Void **IniSetFile**(*sFile*)

Sets default filename for most IniXXX functions.

```
inisetfile("tt.ini");
```

Void **IniSetStr**(*sSec,sName,sStr*)

Writes sName with sStr in section sSec.

```
inisetstr(sect, name, "data");
```

Void **IniSync**(*sFile,iSize*)

Flushes file buffer. The iSize argument sets the size of the RAM buffer.

iSize

-1 Standard buffer size (8k)

0 Normal sync

>0 New buffer size in kilobytes. (Maximum value 63.)

```
inisync("",0);
```

Boolean **IsDir**(*sPath*)

Returns true if path specified is a directory.

```
b = isdir("d:\\century");
```

Integer **LogFile**(*sFile*)

Opens a log file. Returns 0 on success or 1 on failure. Only one log file may be open at a time.

```
log = LogFile("C:\\TT.log");
```

Integer **LogFile_Close**()

Closes the current log file. Returns 0 on success or 1304 on failure.

```
LogFile_Close();
```

Integer **Mkdir**(*sDirName*)

Creates new directory *sDirName* in the current directory. Returns 0 on success and an error code on failure.

```
mkdir("century");
```

String **MkTempNam**(*iFhead, iFext*)

Returns a name for a temporary file.

```
filename = mktempname("temp", "tpx"); // creates file name similar to  
// "temp037e9085412.tmp"
```

Void **Spawn**(*iWait, sCmdString, sArguments*)

Passes *sCmdString* to the operating system to be run with the arguments *sArguments*. Note that the command name, without an extension, must always be the first argument in *sArguments*. Note that you cannot use quotes as part of *sCmdString* or *sArguments*, so if you must launch an application with a space in its Windows filename, you must use DOS 8.3 filenames to refer to that application with the spawn command.

iWait

- 0 Wait until spawned process ends before running next command
- 1 No wait
- 2 Detach spawned process from the console
- 3 Wait for spawned process to complete its startup procedures before continuing with the next command

```
spawn( 0, "notepad.exe", "notepad untitled.txt");
```

String **StdDirectory**(*sStartDir,sCaption*)

Shows standard file open dialog for directories. Returns the full path to the directory chosen.

```
dpath = StdDirectory("d:\\century","TPX FILES");
```

String **StdFont**()

Shows a standard font selection dialog. Returns a comma-delimited string of the font chosen, a zero, and the selected point size.

```
Font = StdFont();
```

String **StdOpen**(*sDir,sMask,sExten,sCaption*);

Opens a Windows standard file browser, returns selected path and filename.

```
filepath = StdOpen("d:\\century","exe files|.exe","exe","Exe Files");
```

String **StdOpenButton**(*sDir, sMask, sExten, sCaption, sButton, iType*)

Opens a Windows standard file browser with an additional button titled *sButton*. The browser dialog box will show an Open or Save button based on the *iType*.

iType

0 Open dialog box

1 Save dialog box

Returns **button* if the user clicks on the additional button, otherwise **StdOpenButton** returns the path and filename the user selected.

```
file = StdOpenButton( "d:\\century", "exe files|.exe|All Files (*.*)|*.*|", "exe",  
                    "Select an Executable","Button Text", 1);
```

String **StdPreview**(*sDir, sTitle*)

Runs **StdOpen** and also displays previews of graphics files.

```
file = StdPreview( "d:\\century", "Images" );
```

Handle **StdPrint**()

Shows a print setup dialog. The return value is a display context handle.

```
hPrint = StdPrint();
```

String **StdSave**(*sPath, sDesc, sExt, sTitle*)

Opens a Windows save browser. Returns the path and filename the user selected, or an empty string if the user chooses Cancel.

```
StdSave( "c:\\temp\\Century", "Text file (*.txt)|*.txt|All Files (*.*)|*.*", "", "Save As" );
```

String **Uniq**(*sFile*)

Tests whether a file can be created. Returns the file name if the file can be created, or a zero-length string if it cannot.

```
filename = Uniq(testfile);
```

Void **WriteLog**(*sMsg*)

Writes text to the log file.

```
WriteLog("Message text.\r\n");
```

USER INTERFACE COMMANDS

Boolean **AddResFile**(*sFileName*)

Adds a resource file to the internal list of files searched for resource data.

```
AddResFile("ttdlgus.res");
```

Handle **CreateForm**(*iFormNum, oParentObject, iDefPage*)

Finds DIALOG or CENTABDLG resource from resource list and creates a modeless dialog whose parent is oParentObject. If the form is a tabbed dialog and iDefPage is nonzero, that page is selected. Returns zero on error.

```
hwnd = CreateForm( 1500, ObjFrame, 0);
```

Void **EnableToolTips**(*bBool*)

Enables or disables tooltips.

```
EnableToolTips(0);
```

Object **GetObject**(*sObjectName*)

Gets the object of string.

```
obj = GetObject("CS103");
```

Object **GetObjectFromHWND**(*hWnd*)

Returns an object from window handle.

```
obj = GetObjectFromHWND(hwnd);
```

Handle **GetObjectHWND**(*oObject*)

Gets hwnd from an object.

```
hwnd=GetObjectHWND(object);
```

Integer **GetObjectID**(*oObject*)

Gets object ID for oObject.

```
IDnum = GetObjectID(object);
```

String **GetObjectName**(*oObject*)

Gets the name of object oObject.

```
oName = GetObjectName(object);
```

Integer **LoadFile**(*sFile, iFlag*)

Loads a *.int file. Returns 0 if the file cannot be found. If iFlag is set to 2, the outer block of the loaded file will be executed.

```
status = LoadFile("test.int",0);
```

String **LoadString**(*iID, iLanguage*)

Gets a string resource from resource list.

```
str = LoadString(200,0);
```

Integer **ModalForm**(*iFormNum, oParentObj, iDefPage*)

Reads DIALOG or CENTABDLG resource and runs modal dialog.

```
nIDCloseButton = ModalForm(1300,DlgObj,0);
```

Void **ObjRegisterFormTooltips**(*hWnd*)

Re-registers a form's tooltips. Normally this occurs automatically.

```
ObjectRegisterFormTooltips(hwnd);
```

Object **ResLoadMenu**(*sStr*)

Finds menu resource from resource list and creates a Win32 menu handle.

```
pmenu = ResLoadMenu("#400");
```

Void **sl_FormExit**(*sObjName, iExitCode*)

Causes a return from ModalForm with iExitCode as the return value.

```
sl_FormExit( DlgObj, IDOK );
```

UTILITY COMMANDS

Void **_Abort**()

Abort script language execution.

```
_Abort();
```

Integer **_Asc**(*sStr*)

Returns ASCII value of the first character of sStr.

```
n = _Asc("Q");
```

String **_Chr**(*iAsciiVal*)

Converts integer ASCII value to single character string.

```
str = _Chr(42);
```

String **_HexStr**(*iInteger*)

Converts integer value to hexadecimal string.

```
hex = _HexStr(42);
```

Boolean **_IsIn**(*sString, iAsciiValue*)

Returns true if the character identified by iAsciiValue is in sString.

```
yes = _IsIn("12345",_asc("3"));
```

Boolean **_IsSpace**(*iAsciiValue*)

Returns true if the character identified by iAsciiValue is a space, tab, carriage return or newline.

```
yes = _IsSpace(_asc(" "));
```

Integer **_SizeOf**(*sStr*)

Returns length of string.

```
n = _SizeOf("12345");
```

String **_Str**(*iVal*)

Returns string from numeric value *iVal*.

```
a = _Str(12);
```

Integer **_StrnCmp**(*sString1*,*iOffset1*,*sString2*,*iOffset2*,*iLen*)

Compares *sString1*, starting at *iOffset1*, with *sString2*, beginning at *iOffset2*, for *iLen* characters. Returns -1 if *sString1* is lesser, 1 if it's greater, or 0 if they're equal.

```
a = _StrnCmp("handle",3,"spindle",4,4);
```

Integer **_StrnCmpi**(*sString1*,*iOffset1*,*sString2*,*iOffset2*,*iLen*)

As `_StrnCmp`, but ignores case.

```
a = _StrnCmpi("HANDLE",3,"spindle",4,4);
```

Void **_ToLower**(*sStr*)

Converts *sStr* to lower case.

```
_ToLower("aBaB");
```

String **_ToLowerC**(*sStr*)

Returns string with all lower case.

```
str = _ToLowerC("aBaB");
```

Void **_ToUpper**(*sStr*)

Converts *sStr* to upper case.

```
_ToUpper("aBaB");
```

String **_ToUpperC**(*sStr*)

Returns string with all upper case.

```
str = _ToUpperC("aBaB");
```

Integer `_Val(sStr)`

Returns integer from a string, octal or hexadecimal number.

```
a = _Val("123");
b = _Val("\033");
c = _Val("0x0ab9");
```

Integer `_VarType(sVarName)`

Returns the type for a named variable.

Return

- 0 Undefined variable
- 1 Integer, Boolean or uninitialized variable
- 2 String
- 17 Integer array
- 18 String array
- 32 System variable

```
vtype = _VarType("file");
```

Time Value `AtoD(sDateTime)`

Converts ASCII time string *sDateTime* to internal time value.

```
tim = AtoD("03/15/1999.01:23:45");
```

String `Cdate(tTime)`

Converts internal time value to ASCII date string.

```
str = cdate(time());
```

String `Chname(iNum,iType)`

Converts ASCII character value to string for display as mnemonic, control or dump.

iType

- 2 Numeric hex values 0d
- 3 Mnemonic ASCII values <CR>
- 4 Control characters ^M
- 5 Keystroke names ENTER

```
str = chname(13,0);
```

String **Ctime**(*tTime*)

Returns clock time string from time value.

```
hour = ctime(time());
```

String **Decrypt**(*sString*)

Decrypts an encrypted string.

```
str = decrypt(password);
```

String **Encrypt**(*sStr*)

Encrypts a passed string.

```
str = encrypt(password);
```

String **Field**(*sList, iPosNum, iSeparator*)

Gets a string from a delineated list.

```
str = field( "a|b|c|d|e", 2, _asc( "|" ) );
```

String **GetEnv**(*sLABEL*)

Gets value of environmental variable.

```
name = getenv("HOME");
```

String **GetSysDir**()

Returns directory TinyTERM is installed in.

```
sdir = GetSysDir();
```

String **GetUserDir**()

Returns user's TinyTERM local directory, if different from system install directory.

```
path = GetUserDir();
```

Void **Help**(*sTopic*)

Runs Windows help system on topic *sTopic*. If *sTopic* is not found or is zero length, the main help window will open.

```
Help("");
```

String **Left**(*sStr*,*iLen*)

Returns leftmost *iLen* characters of string *sStr*.

```
a = left("12345",2);
```

Integer **Main_Exit**(*iRetVal*)

Exit TinyTERM and return *iRetVal*. Must be followed by a Return statement.

```
Main_Exit(2);
```

String **MidStr**(*sStr*,*iPos*,*iLen*)

Returns string starting at *iPos* of length *iLen*.

```
str = MidStr("12345",2,3);
```

Integer **Opsys2**()

Returns an integer that identifies the operating system.

```
OS = opsys2();
```

Integer **Pos**(*sStr1*,*sStr2*,*iStart*)

Returns position of string 1 within string 2, beginning at character number *iStart*. You must set *iStart* to 1 or higher; values less than 1 will always return 0.

```
a = pos("23",11233334445,1);
```

Integer **Quit**(*iExitCode*)

Exits TinyTERM application. Note that in this version of CScript, you must follow quit commands with a return command.

```
Quit(0);
```

```
Return(1);
```

String **Right**(*sStr*,*iLen*)

Returns the rightmost *iLen* characters of *sStr*.

```
a = right("123456",3);
```

Integer **RmDir**(*sPath*)

Removes directory *sPath*. Returns 0 for success, 1 if *sPath* contains wildcard characters, or 1279 for failure.

```
success = RmDir("C:\\Temp");
```

Void **SetUserDir**(*sPath*)

Sets user search directory.

```
SetUserDir("\\Century");
```

Void **Sleep**(*nMilliseconds*)

Pauses for *nMilliseconds*.

```
Sleep( 10000 ); // Pauses execution for 10 seconds
```

String **StrConv**(*sStr*, *iType*)

Converts raw ASCII strings to strings with caret-prefixed control characters and hexadecimal character codes for doublequotes (“\x22”), quotes (“\x27”), and dollar signs (“\x24”). Will also convert converted string back to raw ASCII.

iType

- 0 Convert strings with converted characters to ASCII
- 1 Convert control characters to caret-prefixed characters
- 2 Convert control characters to caret-prefixed characters and
convert double quotes, quotes, and dollar signs to hexadecimal sequences

```
str = strconv("Line one^M^M^MLineTwo^M",0);
```

String **StrHex**(*iInt*,*iLen*)

Gets hex number of length *iLen* from integer *iInt*.

```
a = strhex(12,2);
```

Integer **Sum**(*sString*, *iType*, *iStartVal*)

Calculates integer checksum of string starting with *iStartVal* as the beginning checksum value.

iType

- 0 Adds all characters together
- 1 Xors all characters together
- 2 Returns the CRC checksum of the string

```
i = sum( "string", 0, 0 );
```

Integer **Time**()

Returns current time as an internal time value. This is the number of seconds since 01/01/1970.00:00:00 GMT.

```
tim = time();
```

String **Trim**(*sStr*)

Returns string without trailing spaces.

```
str = trim(s);
```

WIN32 COMMANDS

Void **CheckMenuItem**(*hMenu*, *iItemID*, *iFlag*)

Calls Win32 CheckMenuItem to check or uncheck a menu item.

iFlag

- 0 Uncheck menu item *iItemID*
- 8 Check menu item *iItemID*
- 1024 Uncheck the menu item at the zero-based relative position *iItemID*
- 1032 Check the menu item at the zero-based relative position *iItemID*

```
CheckMenuItem( GetFrameMenu(Frame), 7048, 8 );
```

Void **ContextHelp**(*hWnd*)

Sends an SC_CONTEXTHELP message to the window associated with the object. This will change the mouse cursor to a question mark with a pointer. Clicking on a control after this should open a contextual help window.

```
ContextHelp(hwind);
```

Void **DeleteMenuBarItem**(*hMenu, iItem, iFlag*)

Deletes item number *iItem* from menu *hMenu*. If *iFlag* is set to 0, then *iItem* is the actual menu item number. If *iFlag* is non-zero, *iItem* is the zero-based position of the item in the menu.

```
DeleteMenuBarItem(hmenu,1,1);
```

Void **DestroyMenu**(*hMenu*)

Calls Win32 DestroyMenu function. Used to destroy menu loaded with ResLoadMenu.

```
DestroyMenu(hmenu);
```

Void **DrawMenuBar**(*hWnd*)

Draws the Menu Bar.

```
DrawMenuBar(Frame);
```

Void **EnableMenuItem**(*hMenu, iItem, iTag*)

Calls Win32 EnableMenuItem function.

```
EnableMenuItem(hMenu,700,MF_DISABLED);
```

Void **EnableWindow**(*hWnd, iInt*)

Calls Win32 EnableWindow function.

```
EnableWindow(hwnd,0);
```

Handle **FindWindow**(*sClassName, sWindowName*)

Retrieves a handle to the top-level window that matches the class name and window name specified.

```
FindWindow("dbMon", NULL);
```

String **GetDeviceConfig**(*sDevice*)

Gets configuration information for device *sDevice*.

```
config = GetDeviceConfig(DevName);
```

Object **GetDlgItem**(*hWnd, iSessID*)

Calls Win32 GetDlgItem function.

```
object = GetDlgItem(Frame,1);
```

Object **GetFrameMenu**(*hWnd*)

Returns the frame object's menu handle.

```
hmenu = GetFrameMenu(hFrame);
```

Object **GetSubMenu**(*hWnd, iInt*)

Calls Win32 GetSubMenu function.

```
hmenu = GetSubMenu(hmenu,0);
```

Object **GetSystemMenu**(*hWnd, bBool*)

Calls Win32 GetSystemMenu function.

```
hmenu = GetSystemMenu(Frame,0);
```

Integer **GetSystemMetrics**(*iHW*)

Calls Win32 GetSystemMetrics function.

```
n = GetSystemMetrics(SM_CXSCREEN);
```

String **GetTapiNames**()

Returns names of installed modems.

```
list = GetTapiNames();
```

Void **InsertMenuItem**(*hMenu, iInt1, iItemPos, iMenuItemNum, sLabel*)

Calls Win32 InsertMenuItem function to add a single item to an existing menu.

```
InsertMenuItem(GetSubMenu(GetFrameMenu(Frame,4),3),0,MF_BYPOSITION,ITEM_
NUMBER,"About");
```

Void **InsertMenuParent**(*hMenu, iInt1, iMenuPos, iMenuNum, sLabel*)

Calls Win32 InsertMenu function to add a menu to an existing menu bar.

```
InsertMenuParent(GetFrameMenu(Frame,4),0,MF_BYPOSITION,MENU_NUMBER,"About");
```

Void **MoveWindow**(*hWnd, iX, iY, iW, iH, bRepaint*)

Calls Win32 MoveWindow function.

```
MoveWindow( hFrame, x, y, w, h, 1 );
```

Void **MenuStatus**(*hMenu, iMenuItem, iStatus*)

Changes the status of menu item number *iMenuItem*.

iStatus

- 0 Unchecks the menu item
- 1 Checks the menu item
- 2 Enables the menu item
- 3 Disables the menu item
- 4 Grays the menu item

```
MenuStatus(hmenu,7000,0);
```

Integer **MsgBox**(*sMsg, sCaption, iType*)

Display message box.

iType

- 0 OK button
- 1 OK and Cancel buttons
- 2 Yes and No buttons
- 3 Yes, No and Cancel buttons

```
msgbox("bad entry","ERROR",0);
```

Integer **OpSys2**()

Return operating system type.

- 7 Windows 95, Windows 98 or Windows Me
- 15 Windows 2000 or Windows XP
- 71 Windows NT 4.0

```
os_num = opsys2();
```

Void **RemoveMenuItem**(*hMenu, iMenuItem, iInt*)

Calls Win32 RemoveMenu to remove a menu item.

```
RemoveMenuItem(GetSystemMenu(Frame,0),MENU_DISCONNECT,0);
```

Void **SetDragMode**(*hWnd*)

Capture move and release.

```
SetDragMode(hwnd);
```

Integer **SetErrorMethod**(*iMethod*)

Sets the method used to display error messages. Returns the previous setting.

iMethod

- 0 Use message boxes
- 1 Write to Stderr
- 2 Do not display errors
- 3 Write to the debug monitor

```
previous = SetErrorMethod(0);
```

Boolean **SetFocus**(*hWnd*)

Calls Win32 SetFocus to set window focus to hWnd.

```
target = SetFocus(hWnd);
```

Boolean **SetForegroundWindow**(*hWnd*)

Calls Win32 SetForegroundWindow to bring the window specified in hWnd to the foreground and made the active window.

```
SetForegroundWindow( hWnd );
```

Integer **SetFrameHMENU**(*hWnd, iMenuID*)

Attach a menu to a frame window. Returns the handle of the menu previously attached to the window.

```
SetFrameMenu(hFrame,MenuID);
```

Void **SetFrameMenu**(*hWnd, iMenuResourceID*)

Attach a menu to a frame window. Returns the handle of the menu previously attached to the window.

```
SetFrameMenu(hFrame,MenuID);
```

Boolean **SetHelpFile**(*sFileName*)

Sets name of default help file.

```
SetHelpFile("tt.hlp");
```

Boolean **SetHelpID**(*hWnd, iHelpID*)

Associates a context help ID with a window.

```
success = SetHelpID(hwnd,7748);
```

Void **ShowWindow**(*hWnd, iCmdShow*)

Calls Win32 ShowWindow function.

```
ShowWindow(hwnd,SW_SHOW);
```

Void **TrackPopupMenu**(*hMenu, bBool, iHPos, iVPos, hWnd*)

Calls Win32 TrackPopupMenu function.

```
TrackPopupMenu(GetSubMenu(pmenu,0),0,x,y,Frame);
```

TINYTERM COMMANDS

Void **AppRedraw**()

Redraws application user interface.

```
AppRedraw();
```

Void **BrowseToURL**(*sURL*)

Runs the default Internet browser and connects to the specified URL.

```
BrowseToURL("http://www.centurysoftware.com");
```

Boolean **CompileFile**(*sFilename*)

Loads and executes a CScript file. Returns true if the file ran successfully.

```
success = CompileFile("jackpot.cs");
```

Integer **CompileString**(*sCommand*)

Executes a CScript command string. Returns 1 if the command had no errors or -1 if there were errors.

```
status = CompileString("count++");
```

String **ConfigureDevice**(*iIndex, sDeviceName, hWnd*)

Gets modem configuration.

```
dconfig = ConfigureDevice(1,cdevice,HWND);
```

String **CSLdebug**(*sArg*)

Returns CScript debugging information based on argument sArg.

sArg

- ? List all other arguments
- q Quit and exit
- g Go
- t [n] Trace n steps
- l fname Load file fname
 - d Dump symbol dictionary
 - d sym Dump symbol sym. Can be wildcarded.
 - d n cnt Dump stack value n
 - bp List break points
 - bp sym Set break point on symbol sym
 - bd sym Delete break point on symbol sym
 - bd * Delete all break points
 - st Stack trace
 - r Register dump
 - u sym Unassemble code for symbol sym
 - . Display current instruction (not enabled)

```
symdic = CSLdebug("d");
```

String **GenerateURL**(*sSystem, sUsername, sPassword, sStr, iInt, iInt2, iInt3*)

Generates URL using operator entries.

```
str = GenerateURL("www.censoft.com", "anonymous", pword, "", 0, 0, 0);
```

String **GetCountryList**()

Gets countries supported by modem location list.

```
list = GetCountryList();
```

String **GetDefaultBrowser()**

Gets default browser name and path.

```
filename = GetDefaultBrowser();
```

String **GetDefaultBrowserAppName()**

Gets only default browser filename.

```
file = GetDefaultBrowserAppName();
```

String **GetLocationList()**

Gets locations supported by modem.

```
list = GetLocationList();
```

Integer **GetPropInt(*oObject*,*sPropertyName*)**

Returns the integer value of property *sPropertyName* associated with object *oObject*. Returns 0 if the object or property cannot be found.

```
propval = GetPropInt(hwnd,propname);
```

String **GetPropStr(*oObject*,*sPropertyName*)**

Returns the string value of property *sPropertyName* associated with object *oObject*. Returns a zero-length string if the object or property is not found.

```
propval = GetPropInt(hwnd,propname);
```

Boolean **IsDebugMode()**

Returns true if “-debug” is set in command line.

```
b = IsDebugMode();
```

Integer **IsSess(*oSession*)**

Returns session number for object *oSession*.

```
sessnum = IsSess(CSESS);
```

Void **KeyLoad(*sKeyFile*,*sSection*,*iFlag*)**

Loads keyboard definition *sSection* from keyboard file *sKeyFile*. *iFlag* is not used.

```
KeyLoad(“keyboard.dat”,“Default.keyboard”,0);
```

Integer **KeySave**(*sKeyFile,sSection*)

Saves the current keyboard mappings to sSection of sKeyFile. Returns 0 on success or -1 on failure.

```
KeySave("keyboard.dat","New section.keyboard");
```

Void **Menu_Draw**()

Redraws the Menu Bar if enabled.

```
menu_draw();
```

Integer **PerformLicenseCheck**(*sProdCode,iBitmask,iVersion,bErrMsg*)

Tests whether a Century product is licensed. The version is always 1024 for TinyTERM version 4. If bErrMsg is set, a message window will pop up if there are any errors. The valid product codes for TinyTERM 4.20 and 4.21 are:

sProdCode

EV3	TinyTERM Emulator
PL3	TinyTERM Plus Edition
TC3	TinyTERM Thin Client Edition
WS3	TinyTERM Web Server Edition

The possible return codes are:

iRetVal

0	Invalid license
1	Valid license
2	Expiration reminder needed
4	Expired evaluation license
8	No license found

```
menu_draw();
```

Void **Script_Login**()

Callback function called after login completes. Note: when using any of the Script_ commands, you should avoid using dialog boxes if you will be running from within a browser.

```
script_login();
```

Void **Script_Logout()**

Callback function called before logout begins.

```
script_logout();
```

Void **Script_SessConnect()**

Callback function called after connecting to the host.

```
script_sessConnect();
```

Void **Script_SessDiscon()**

Callback function called before disconnecting from the host.

```
script_sessDiscon();
```

Void **Script_SessDown()**

Callback function called before the session is closed.

```
script_sessdown();
```

Void **Script_SessUp()**

Callback function called after the session is opened.

```
script_sessup();
```

Void **Script_ShutDown()**

Callback function called before shutdown.

```
script_shutdown();
```

Void **Script_StartUp()**

Callback function called after startup.

```
script_startup();
```

Void **SessDel(oSession)**

Deletes session with object oSession.

```
SessDel( CSESS );
```

Void **SessionNew**(*sTpxFile*);

Starts a new session.

```
SessionNew("c:\\tmp\\default.tpx");
```

Void **Setkey_Reset**(*iFlag*)

Resets all keyboard macros. If iFlag is set to 0, all emulator keys will be reset as well.

```
Setkey_Reset(1);
```

Void **StatusBar_Draw**()

Redraws Status Bar if enabled.

```
StatusBar_Draw();
```

Void **SwitchSess**(*nSessNum*,*iSave*)

Switches to session nSessNum. iSave is currently non-functional.

```
SwitchSess(3,0);
```

JAVASCRIPT COMMANDS

break;

Exit current command loop and continue script past its end.

```
break;
```

continue;

Restarts loop processing from the first command.

```
continue;
```

do { commands } while (condition);

Execute commands until condition is met. The commands will always be executed at least once.

```
i = 0;
```

```
do {
```

```
    i++;
```

```
} while ( i < 10);
```

for (iVar; condition; operation) { commands }

Execute commands until condition is met.

```
for (i = 1; i < 15; i * 2) { dprintln(_str(i)); }
```

function fname(variables) { commands }

Creates a function named fname. If variables are named, they must be passed to the function. CSL functions can be called recursively. Excessive recursive calls can result in a stack overflow.

```
Function example(sTitle) {  
    msgbox("The function works",sTitle,0);  
}  
example("My title");
```

if (condition) { commands1 } else { commands2 }

Conditionally executes commands. Else is not required.

```
if (i == 4) {  
    msgbox("i = 4","Value",0);  
}
```

var vName

Initialize variable vName. Multiple variables may be initialized with one command.

```
var sTitle,iCount;
```

while (condition) { commands }

As Do, but commands will be executed only if condition is met.

```
i = 4;  
while (i < 3) { i++;}
```

C COMMANDS

goto LABEL

Continue script execution at LABEL. The label must be followed by a colon.

```
goto NEW_ZONE;
```

```
NEW_ZONE :
```

```
...
```

Switch (iValue) { Case ... Default }

Executes one set of commands based on iValue.

```
switch (time()%2) {
```

```
    case 0 :
```

```
        te.DisplayNL("Time is even.");
```

```
        break;
```

```
    case 1 :
```

```
        te.DisplayNL("Time is odd.");
```

```
        break;
```

```
    default :
```

```
        te.DisplayNL("Error in modulo operation.");
```

```
}
```

DDE COMMANDS

DDE SERVER COMMANDS

Void **DDE_Enable**(*bOnOff*)

Enables or disables the DDE server.

```
DDE_Enable(1);
```

Void **DDE_Name**(*sName*)

Sets the DDE server name.

```
DDE_Name("TINYTERM");
```

Void **DDE_TimeOut**(*iInterval*)

Sets the DDE client/server timeout in milliseconds.

```
DDE_TimeOut(5000);
```

DDE CLIENT COMMANDS

Integer **DDE_Advise**(*iChannel,sItem,sVar*)

Requests an advise link on sItem from the DDE server application and stores the result in sVar. Returns 0 on success, 1 on failure, or 420 if iChannel is not open.

```
result = DDE_Advise(1,"Name","sName");
```

Void **DDE_Close**(*iChannel*)

Closes a DDE channel.

```
DDE_Close(1);
```

Void **DDE_CloseAll**()

Closes all DDE channels.

```
DDE_CloseAll();
```

Integer **DDE_Execute**(*iChannel,sTransaction*)

Executes a DDE transaction. Returns 0 for success, 1 for failure, or 420 if iChannel is not open.

```
DDE_Execute(1,CmdStr);
```

Integer **DDE_Init**(*iChannel,sService,sTopic*)

Opens a DDE channel. Returns 0 on success or 1 on failure.

```
result = DDE_Init(1,sApp,AppTopic);
```

Integer **DDE_Poke**(*iChannel,sItem,sData*)

Sends data to the DDE server. Returns 0 on success, 1 on failure or 420 if iChannel is not open.

```
DDE_Poke(1,AppItem,DataString);
```

Integer **DDE_Request**(*iChannel,sItem,sVar*)

Requests data from the DDE server and stores it in sVar. Returns 0 on success, 1 on failure or 420 if iChannel is not open.

```
DDE_CloseAll();
```

Integer **DDE_Unadvise**(*iChannel,sItem*)

Closes the advise link on sItem in a DDE conversation. Returns 0 for success, 1 for failure, or 420 if iChannel is not open.

```
DDE_Unadvise(1,AppItem);
```

Chapter 3

TE Control Object Reference

In this chapter

Properties

Function Reference

Events Reference

TE functionality includes terminal emulation, communications relating to terminal emulation, host mode (server), and all properties and functions relating to the above capabilities. The TE control does not include any file-transfer components.

The following CScript demonstrates some of the TE Object's features and how to use them. This script asks the user for the host name that the user wants to connect to, sets the necessary properties for a default Telnet connection, and connects to the host the user entered.

```
// get hostname from the user
var node = te.Read("Hostname: ", 1);

// set TE properties and connect
te.Node = node;           // Sets node = "Hostname"
te.NetPort = 23;         // Default Telnet Port
te.ConnectionType = 0;   // Telnet connection
te.Emulation = 5;       // Set emulation type to SCOANSI
te.AutoLogin = 0;       // Turns autologin off
te.Connect();           // Connects to host
```

Properties

Object properties act as variables. To set the value of a property, use the format `te.Property = argument`. For example:

```
te.Emulation = 0;
```

To get the value of a property, use the property as the argument of an assignment or wherever you need the property's value. For example:

```
var sNode = te.Node;
```

Each TE object includes the following properties:

Property Name	Data Type	Description
AddCR	Boolean	Add CR to end of line. Default = FALSE
AddLF	Boolean	Add LF to end of line. Default = FALSE
AutoConnect	Boolean	Connect on session open. Default = FALSE
AutoLogin	Boolean	Login after connect. Default = FALSE
BackspaceIsDelete	Boolean	Default = TRUE
Baud	Integer	Asynch communication baud rate. Default = 9600
Blockmode	Boolean	Default = FALSE
CanAbort	Boolean	Allow ^C interruption for some functions. Default = FALSE
CaptureDevice	String	Device or file for data capture. Default = capt#.fil
CaptureFlush	Boolean	Close capture file at end of data. Default = FALSE
CaptureMode	Integer	0 ASCII (default) 1 BINARY 2 CONTROL 3 MNEMONIC
CaptureOn	Boolean	Data capture setting. Default = FALSE
CaptureOverwrite	Boolean	Overwrite existing capture file. Default = FALSE (append)
CaptureType	Integer	0 PRINT MANAGER 1 DEVICE 2 FILE (default) 3 SPOOL

continued...

continued from previous page...

CharSet	Integer	0	American (Default)
		1	British
		2	Dutch
		3	Finnish
		4	French
		5	Canadian
		6	German
		7	Italian
		8	Danish
		9	Norwegian
		10	Portugese
		11	Spanish
		12	Swedish
		13	Swiss
ChiseledLineDraw	Boolean	Use thin graphical line draw. Supersedes SolidLineDraw. Default = FALSE	
ChiseledSelection	Boolean	Indented text selection. Default = FALSE	
CommLink	String	Name of COMM object to load. Default = COMM.COM	
ComPort	Integer	Sets the COM port used for asynchronous communications by number	
ConnectionType	Integer	0	Telnet (Default)
		1	Rlogin
		2	Asynchronous
		3	TAPI
		4	SSL/TLS
		5	SSGH
CursorType	Integer	0	Blinking Block (Default)
		1	Solid Block
		2	Blinking Line
		3	Solid Line
DestructiveBackspace	Boolean	Backspace blanks out Characters. Default = FALSE	
DisconnectOnExit	Boolean	Default = TRUE	
DropDTROnExit	Boolean	Default = TRUE	
DuplexMode	Integer	0	Full (Default)
		1	Half
		2	Mnemonic
		3	Control
Emulation	Integer		

Note that the value you use to set the emulation mode and the value you get when you get the emulation do not correspond.

<u>Set Value</u>		<u>Get Value</u>	
UNIX Modes		UNIX Modes	
0	ADM1	0	
1	ANSI	1	VT320
2	AT386	2	VT320-7
3	IBM 3101	3	VT220
4	PCTERM	4	VT220-7
5	SCOANSI (Default)	5	VT100/VT102
6	TTY	6	
7	TV912	7	ANSI
8	TV925	8	AT386
9	TV950	9	SCOANSI
10	VT100	10	WYSE60
11	VT102	11	WYSE60-25
12	VT220	12	PCTERM
13	VT220-7	13	TV950
14	VT320	14	WYSE50
15	VT320-7	15	TV925
16	WYSE50	16	TV912
17	WYSE60	17	ADM1
18	WYSE60-25	18	IBM 3101
19	IBM 3151	19	IBM 3151
20	IBM 3151-25	20	IBM 3151-25
		21	TTY
TN5250 Modes			
21	IBM3179-2	21	IBM3179-2
22	IBM3477-FT	22	IBM3477-FT
TN3270 Modes			
23	IBM3278-2	23	IBM3278-2
24	IBM3278-3	24	IBM3278-3
25	IBM3278-4	25	IBM3278-4
26	IBM3278-5	26	IBM3278-5
ESCPassThru	Boolean		Telnet ESC pass-through. Default = FALSE
Fingerprint	String		Current SSH fingerprint.
HasSSH	Boolean		True if current session is SSH. Read-only.
IgnoreHostColor	Boolean		Disables processing of sequences sent from host. Default = FALSE

IgnorePrintStop	Boolean	Pause printing until PrintStopFlags condition is met. Default = FALSE
IsConnected	Boolean	TRUE if a connection is active. Read-only.
JumpScroll	Integer	Number of lines to jump scroll. Set to 0 (default) to turn off
KeyMacroPse	Boolean	Pause the Keyboard Macro Recorder. Default = FALSE
LastError	Integer	Number of the last error. Read-only.
LDelay	Integer	Milliseconds of pause after a linefeed. Default = 0
LoginString	String	String that controls auto login.
LogoffString	String	String that controls auto logout.
MaskParity	Boolean	Enables stripping of the high bit from incoming data. Default = FALSE
NetPort	Integer	Sets the TCP/IP port for Rlogin, Telnet or SSH connections. Defaults Telnet 23 Rlogin 513 SSH 22
Node	String	Specifies hostname, IP address, or phone number for Telnet, Rlogin, or TAPI connections
NoScroll	Boolean	Controls scrolling. Default = FALSE
NumColumns	Integer	1-132. Default = 80
NumLines	Integer	1-66. Default = 25
NumLockON	Boolean	Default = FALSE
Pages	Integer	Number of screen pages, 1-6. Default = 6

Parity	Integer	Asynch parity 0 None (Default) 1 Odd 2 Even 3 Mark 4 Space
Password	String	Password for auto-login
PrinterColumns	Integer	Number of columns on a printed page. Default = 0 (auto set)
PrinterFile	String	File or device for printing.
PrinterFlags	Integer	For printing to file. 0 Append (default) 1 Overwrite
PrinterFlush	Boolean	Flush print buffer at end of job. Default = FALSE
PrinterLines	Integer	Number of lines on a printed page. Default = 0 (auto set)
PrinterMode	Integer	Determines printer setup 17 Print to file 18 Direct to device 19 Ignore print requests 20 To Windows printer
<p>Add 32 to the number above to bypass the Windows printer driver. (Only affects setting 20.) Add 128 to add a form feed to the end of the print job. (Does not affect settings 17 or 19.)</p>		
PrinterPause	Integer	Number of seconds with no new data to wait before printing. Default = 5
PrintStopFlags	Integer	1 Print on key press 2 Print on timeout
ProxyHostID	String	Hostname or IP address of the proxy server.
ProxyPassword	String	Password for a proxy connection.
ProxyPort	Integer	TCP/IP port number for a proxy connection.
Proxytn_gw_nav	Boolean	Sets tn_gw_nav for proxy connections. Default = FALSE

ProxyType	Integer	0 None 1 Generic 2 SOCKS 4 3 SOCKS 5, no authentication 4 SOCKS 5, username & password
ProxyUserID	String	Username for a proxy connection.
RTS	Boolean	RTS/CTS flow control. Default = FALSE
ScrollBack	Integer	Number of lines to store in the scrollbar buffer. Default = 999
ScrollBar	Boolean	Enables scrollbar. Default = FALSE
ShowTelnet	Boolean	Show telnet negotiation. Default = FALSE
SolidLineDraw	Boolean	Use thick graphical line draw. Superseded by ChiseledLineDraw. Default = FALSE
SSHCompressionLevel	Integer	A number from 0 (none) to 9 (highest, default)
SSHCypherType	Integer	0 None 1 DES 2 3DES (default) 3 Blowfish
StopBits	Integer	Stop bits for asynch communication. Default = 1
TabEx	Boolean	Expand tabs. Default = TRUE
TDelay	Integer	Milliseconds of pause after a tab character. Default = 0
TermID	String	VT ID of Terminal
TurnCharacter	Integer	ASCII value of IBM turnaround character. Default = 13
UseAlt	Boolean	Enables handling of ALT keys in emulators. Default = FALSE

UserName	String	User name for auto-login
ViewLogin	Boolean	View auto-login process. Default = FALSE
WaitForCall	Boolean	Waits for an incoming call on the modem. Default = FALSE
WindowsLookandFeel	Boolean	Enables Windows Look and Feel mode, a combination of ChiseledSelection and ChiseledLineDraw. Default = FALSE
WordLength	Integer	Asynch communication word length. Default = 8
WrapLines	Boolean	Wrap long lines. Default = TRUE
WRU	String	String sent after WRUChar is received.
WRUChar	Integer	ASCII value of character to request WRU string. Default = 5
XonCom	Boolean	XON/XOFF flow control. Default = FALSE

Function Reference

TE objects are accessed in scripting with the format `te.ObjectName(parameters)`. For example, to clear the screen, the command is:

```
te.cls();
```

Integer **AddFont**(*sFontName,sCodePage,iFontIndex*)

Add a font to the font list in the TE control. *sCodePage* is limited to the short name of the code page. The code page file is now hardcoded to be `tcs.dat`. *iFontIndex* runs from 0-(MAXFONTS-1), where MAXFONTS is currently set to 4, and is a compile-time #define. Passing -1 to *iFontIndex* will load the font in the next available slot. Returns -1 on failure, and the index where the font was loaded on success.

```
index = te.AddFont("Arial",1252,-1);
```

Void **Break**()

Send a Break signal to the host.

```
te.Break();
```

Void **CancelDialog()**

Cancel a currently running dialog string.

```
te.CancelDialog();
```

Boolean **Capture**(*nType,sFile*)

Starts data capture of type nType (as CaptureType above) to file or device sFile. Returns 0 on successful capture start.

```
te.Capture(2,"C:\\capt#.fil");
```

String **Chrdy()**

Returns the character pressed at the keyboard as a string, without waiting. If no key was hit, returns a zero-length string.

```
letter = te.Chrdy();
```

String **Chrin()**

Waits for a character to be pressed at the keyboard and returns it as a string.

```
letter = te.Chrin();
```

Integer **Cls()**

This function will clear the terminal emulator screen, as well as the scrollbar buffer. Always returns 0.

```
te.Cls();
```

String **ComIn()**

Looks for a character to be received at the communications line and returns it as a string. If no character is received, returns a zero-length string.

```
ltr = te.ComIn();
```

Boolean **ComSt()**

Returns true if a character is received at the communications line.

```
code = te.ComSt ();
```

Integer Connect()

Attempts to make a connection to a remote server, using the parameters set earlier. If this method succeeds (indicated by the return code) an EConnect (for a successful connection) or an EDisconnect (for a failed connection) event will eventually be fired. The success of this function merely indicates that a connection is in progress, not that we are actually connected with the remote server. For instance, a TELNET connection to a remote machine will return immediately (assuming that all the parameters are set correctly), but the connection is not actually established until the EConnect is received. On failure of the connection, EDisconnect will be fired. Returns 0 for success, non-zero for various failure modes, depending upon the type of failed connection.

```
mode = te.Connect();
```

Void CopySelection(*x,y,w,h*)

Copy a portion of the emulator screen to the clipboard in text format. The top left corner of the emulator is $x=1$ and $y=1$. If all parameters are 0, copies the currently selected portion of the emulator screen to the clipboard. If all parameters are -1, the entire emulator screen will be copied.

```
te.CopySelection(2,1,15,1);
```

String Cread(*iTimeout,iChars,bOption*)

Reads characters from the communications line into a string variable. Writes to the variable after $nChars$ characters are received, or after $nTimeout$ seconds have passed. If $nTimeout$ is 0, it will not time out.

Option

- 0 No processing
- 1 Process incoming text through `te.Trans()`

```
msg = te.Cread(0,15,0);
```

Integer CreadInt(*iTimeout,iChars,bOption*)

Reads characters from the communications line into an integer variable. Acts as `Cread()` in all other respects.

```
count = te.Creadint(0,15,0);
```

Boolean DeleteFont(*iFontIndex*)

Remove a font entry from the font table. As the font list in TE is not a sparse array, the font list is compacted, and the fonts above the index to be deleted (if any) are moved down, decrementing their index values.

```
te.DeleteFont(2);
```

Boolean **Disconnect**()

Disconnect from the current connection. If this function succeeds, an EDisconnect event will be fired. Disconnecting when there is no current connection is considered a failure.

```
te.Disconnect();
```

Void **Display**(*sDisplay*)

Output the given string on the terminal emulator display at the current cursor position, without sending it to the CommLink or appending a newline.

```
te.Display("Hello world");
```

Void **DisplayNL**(*sDisplay*)

Output the given string on the terminal emulator display at the current cursor position and append a newline.

```
te.DisplayNL("Hello world");
```

Integer **DoDialog**(*sString*)

Execute a dialog string. Returns 0 on success, -1 for a timeout, and -2 for a user cancel.

```
dcode = te.DoDialog(instr);
```

Void **Flush**()

Flush the communications buffer.

```
te.Flush();
```

Integer **GetBGColor**(*iAttr*)

Return the current background color of text attribute *iAttr*.

Attribute

- 0 Normal
- 1 Reverse
- 2 Blink
- 3 Underline
- 4 Bold
- 5 Reverse Blink
- 6 Reverse Underline
- 7 Reverse Bold
- 8 Blink Underline
- 9 Blink Bold
- 10 Underline Bold
- 11 Reverse Blink Underline
- 12 Reverse Blink Bold
- 13 Reverse Underline Bold
- 14 Blink Underline Bold
- 15 Reverse Blink Underline Bold

Color

- 0 Default, based on Normal attribute
- 1 Black
- 2 Blue
- 3 Green
- 4 Cyan
- 5 Red
- 6 Magenta
- 7 Brown
- 8 White

```
BGcolnum = te.GetBGColor(0);
```

Integer **GetFGColor**(*iAttr*)

Return the current foreground color of text attribute iAttr.

Color

- 0 Default, based on iAttr 0
- 1 Black
- 2 Blue
- 3 Green
- 4 Cyan
- 5 Red
- 6 Magenta
- 7 Brown
- 8 White
- 9 Light Gray
- 10 Light Blue
- 11 Light Green
- 12 Light Cyan
- 13 Light Red
- 14 Light Magenta
- 15 Yellow
- 16 Light White

```
FGcolnum = te.GetFGColor(0);
```

String **GetFont**(*iIndex*)

Return the name, width in pixels and height in pixels of the font loaded in zero-based position iIndex.

```
font2 = te.GetFont(1);
```

Integer **GetRGBEntry**(*iEntry*)

Return the current RGB value from the index specified by iEntry from the RGBMap.

```
color = te.GetRGBEntry(ccode);
```

String **GetSelection**(*x,y,w,h,iType*)

Copy a portion of the emulator screen and return it. The x, y, w, h parameters behave exactly like Copy-Selection. Only iType 0 is supported, meaning that only strings will be returned.

```
appname = te.GetSelection(2,1,15,1,0);
```

Integer **GetTextAttribute**(*iAttr*)

Returns a bitfield indicating which characteristics are applied to text attribute *iAttr*. The bitfield will be a sum of the following:

Characteristic

- 1 Bold
- 2 Underline
- 4 Blink
- 8 High bright
- 16 Low bright

```
applied = te.GetTextAttribute(0);
```

Boolean **IsClipboardEmpty**()

Returns true if nothing is stored in the clipboard.

```
check = te.IsClipboardEmpty();
```

Integer **KeyIn**()

Returns the number of the key pressed.

```
keynum = te.KeyIn();
```

Void **KeyMacroEnd**()

Stops the Keyboard Macro Recorder.

```
te.KeyMacroEnd();
```

String **KeyMacroRec**(*sFile*)

Starts the Keyboard Macro Recorder and stores the results in the file specified by *iFile*. If *iFile* includes the # symbol, that will be replaced by a two-digit number. Returns the filename used.

```
macrofile = te.KeyMacroRec("macro#.cs");
```

Boolean **LoadKeyboard**(*sKeyFile*,*sLayout*)

Load keyboard layout *sLayout* from the .dat file specified by *sKeyFile*.

```
te.LoadKeyboard("keyboard.dat","English");
```

Void **MouseFunction**(*iButton, iFunction, sMouseString*)

Program mouse button *iButton* to perform *iFunction* using *sMouseString*. Valid mouse strings are listed in the TinyTERM Help file.

Button

- 0 Left
- 1 Middle
- 2 Right

Function

- 0 Disable
- 1 Select
- 2 Send custom string

```
te.MouseFunction(1,2,"%o");
```

Void **Paste**(*iType*)

Paste the contents of the clipboard into the emulator. *iType* 0 means to paste text, and 1 is link format. The data pasted is not processed by the keyboard code pages, but rather is sent out the comm line (if any) without additional processing except for carriage return/line feed corrections and duplex settings.

```
te.Paste(0);
```

Void **PrintScreen**()

Print the current emulator screen.

```
te.PrintScreen();
```

Void **PrintSetup**(*hWndParent*)

Display the page setup dialog to allow customization of the print settings.

```
te.PrintSetup();
```

String **Read**(*sDisplay, bEcho*)

Display prompt *sDisplay* and wait for user input, returning the string the user entered once RETURN (or ^C if *CanAbort* is set) is pressed. Characters entered by the user are not sent on the *CommLink*, but are echoed to the emulator display if *bEcho* is set to TRUE.

```
uname = te.Read("Enter username: ",false);
```

Void **Redraw**()

Redraws the emulator screen.

```
te.Redraw();
```

Void **RevDim**(*bOnoff*)

Determines whether Wyse60 reverse fields are displayed as dim.

```
te.RevDim(1);
```

Integer **SetAltCP**(*sDatafile,sCodepage,iAltpagenum*)

Sets the code pages for the SCOANSI alternate character set. *iAltpagenum* can be 1 or 2.

Returns 0 for success.

```
te.SetAltCP("C:\\Program Files\\Century\\TinyTERM\\codepage.dat","STD 437 MS-DOS Latin US",2);
```

Boolean **SetBackgroundImage**(*sImage,iMethod*)

Display background image *sImage* in the emulator window.

iMethod

- 0 Stretch image to fit the size of the terminal emulator
- 1 Tile image inside the terminal emulator
- 2 Center the image inside the terminal emulator

```
te.SetBackgroundImage("c:\\images\\walpapr.jpg",2);
```

Void **SetBGColor**(*iAttr,iColor*)

Set background color for text attribute *iAttr* to color number *iColor*. Uses the same attribute and color numbers as `GetBGColor`.

```
SetBGColor(4,7);
```

Boolean **SetDECMNCP**(*sDATFile,sCodePage*)

Load VT220/VT320 multinational codepage *sCodePage* from .dat file *sDATFile*.

```
mode = te.SetDECMNCP(CPfile,"1252");
```

Boolean **SetDECSGCP**(*sDATFile,sCodePage*)

Load VT220/VT320 special graphics codepage *sCodePage* from .dat file *sDATFile*.

```
mode = te.SetDECSGCP(CPfile,"1252");
```

Void **SetDTRExit**(*bNewValue*)

Drop DTR on exit.

```
te.SetDTRExit(true);
```

Boolean **SetEmInCP**(*sDATFile,sCodePage*)

Load receive codepage *sCodePage* from .dat file *sDATFile*.

```
mode = te.SetEmInCP(CPfile,"1252");
```

Boolean **SetEmOutCP**(*sDATFile,sCodePage*)

Load transmit codepage *sCodePage* from .dat file *sDATFile*.

```
mode = te.SetEmOutCP(CPfile,"1252");
```

Void **SetFGColor**(*iAttr,iColor*)

Set foreground color for text attribute *iAttr* to color number *iColor*. Uses the same attribute and color numbers as `GetFGColor`.

```
SetBGColor(4,11);
```

Boolean **SetFontCP**(*iFontIndex,sCPDataFile,sCodePage*)

Select new code page *sCodePage* from file *sCPDataFile* for font *iFontIndex*.

```
te.SetFontCP(2,CPfile,"1252");
```

Boolean **SetKbdCP**(*sDATFile,sCodePage*)

Load keyboard codepage *sCodePage* from .dat file *sDATFile*.

```
mode = te.SetKbdCP(CPfile,"1252");
```

Void **SetKey**(*iKey,iType,sValue*)

Set key number *iKey* using mode *iMode* to send *sValue*. *iMode* should always be set to -1. To send a script command, add <COMMAND> to the beginning of the string.

```
te.SetKey(609,-1,"<COMMAND>te.displaynl("Hello there!");");
```

Void **SetRGBEntry**(*iEntry,iRGBValue*)

Modify entry number *iEntry* in the RGBMap.

```
te.SetRGBEntry(2,newcolor);
```

Void **SetTextAttribute**(*iAttr,iChar*)

Sets characteristics of text attribute *iAttr* according to bitfield *iChar*.

```
te.SetTextAttribute(0,12);
```

Boolean **SetVT100SGCP**(*sDATFile,sCodePage*)

Load VT100 special graphics codepage *sCodePage* from .dat file *sDATFile*.

```
mode = te.SetVT100SGCP(CPfile,"1252");
```

Boolean **StopPrint**()

Cancel a print job or print capture.

```
te.StopPrint();
```

Void **Terminal**(*sString*)

Pause script execution until *sString* is received from host, or until the user types Ctrl-E.

```
te.Terminal("$$$");
```

Void **Trans**(*iType,iSource,iDest*)

Translate a specified character received or transmitted to a different character value. *iSource* is the ASCII value of the character to translate, and *iDest* is the ASCII value of the resulting character.

```
te.Trans(1,65,97);
```

Integer **Wait**(*sString,iTimeout*)

Wait *iTimeout* seconds for string *sString* to be received from the commline, or for the user to cancel by typing Ctrl-C. If *iTimeout* is set to 0, Wait will not time out. Returns 0 on success, -1 for a timeout, and -2 for a user cancel.

```
success = te.Wait("word: ",5);
```

Multiple match strings may be specified by separating them with the characters “^S|^S”. If this is done, the return value will be the zero-based position of the matched string. For example,

```
ret_val = te.Wait(“word^S|^Sclient”,5);
```

will return 0 if “word” is received, and 1 if “client” is received.

Integer **Xmit**(*sString*)

Send string *sString* over the current connection. The string is not processed by the keyboard code page, but is processed with carriage-return/linefeed translations and duplex mode. Returns an integer representing the number of characters transmitted, or -1 on error.

```
numchars = te.Xmit(uname);
```

Events Reference

Click()

This event fires when a click is performed in the control.

DbClick()

This event fires when a double click is performed in the control.

EBinaryMode()

This event fires when the emulator is set to binary mode.

ECapsLockOff()

This event fires when Caps Lock is turned off.

ECapsLockOn()

This event fires when Caps Lock is turned on.

ECapture(bON,iCaptureType,sDevice)

This event fires when a capture operation starts or stops. *bON* is TRUE when capture is being turned on, FALSE when it is being turned off. *iCaptureType* is the type of capture that is currently configured. *sDevice* is the device that is being used for capture.

EConnect()

This event fires when a connection has occurred successfully.

EDisconnect()

This event fires when a connection has terminated (either through user action or a remote disconnection; i.e., when a socket is disconnected). This event also fires when an attempted connection fails. You will not receive this event if a given TE control is destroyed before the connection is terminated.

EExit()

This event fires when the user has pressed ALT-F4. This can only happen when the UseAlt property is set to FALSE. This can be used by the application to shut down.

ELoginStart()

This event fires just before a login or logout scheme is performed.

ELoginStop()

This event fires just after a login or logout scheme has been performed.

ENextSession()

This event fires when the user has indicated (via a keystroke) that they want to advance to the next session. The actual switching of sessions must be handled by the application.

ENumLockOff()

This event fires when Num Lock is turned off.

ENumLockOn()

This event fires when Num Lock is turned on.

EPrint(bON)

The event fires when printing commences or stops. bON is set to TRUE when printing starts, and FALSE when printing ends.

MouseDown(iButton,iShift,x,y)

This event fires when a mouse button is depressed in the control. *iButton* is the number of the mouse button that is depressed. The left button is 1, the right button is 2, and the middle button is 4. *iShift* is the status of the shifting keys at the time of the keypress. *iShift* is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. *iShift* indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *iShift* would be 6. *x* and *y* are the *x* and *y* positions of the mouse cursor, relative to the upper left hand corner of the control.

MouseUp(iButton,iShift,x,y)

This event fires when a mouse button is released in the control. The arguments are identical to those for *MouseDown*.

Chapter 4

FT Control Object Function Reference

In this chapter

File Transfer Session Management

Properties

Function Reference

Events Reference

Sample File Transfer Script

This reference is intended to be a guide to the functions that constitute the main interface to the File Transfer functionality currently contained within TinyTERM user interface.

These functions constitute the public file transfer interface. This permits the sending and receiving of files, and the setting and getting of a limited set of file transfer properties. Aside from filenames and directories, all transfers are configured from the existing properties in the current session's .TPX file. This includes whether or not transfer status screens are displayed, whether or not TinyTERM should exit when a transfer is complete and all other options that are able to be set from the User Interface, .TPX file or programmatically. All the property set and get functions operate on these properties, meaning that if they are changed programmatically or through the user interface, they will update the file transfer dialogs when they are next used and if the .TPX file is saved, these properties will be saved into it.

File Transfer Session Management

File transfer maintains its own list of FT sessions. Most of these sessions are associated with emulator sessions, with the exception of FTP file transfers, which are never linked to an emulator session. Only one file transfer can be run per session, with the above noted exception of FTP. The maximum number of file transfers (FTP and otherwise) is currently 20, and is defined as twice the number of maximum TE sessions (currently 9) plus 1.)

Properties

GET PROPERTIES

str FTGetRemoteDir()

This function returns the currently set remote directory to be used for file sends. It is retrieved from the first field of the list contained in property 214 FTDefSend. (The properties are defined later in this document.)

str FTGetUserName()

This function returns the user name as a string that is to be used for FTP file transfers. It takes into account the setting of property 264 FTPLogType, (properties are defined later in this document.) meaning that it will return different values based on the setting of this property.

str FTGetPassword()

This function returns the decrypted password as a string that is to be used for FTP file transfers. It takes into account the setting of property 264 FTPLogType, (properties are defined later in this document.) meaning that it will return different values based on the setting of this property.

str FTGetHostName()

This function returns the host name as a string that is to be used for FTP file transfers. It takes into account the setting of property 349 UseFTPCurHost, (properties are defined later in this document.) meaning that it will return different values based on the setting of this property.

int FTGetProtocolID()

This function returns the zero-based file transfer protocol ID from property 99 xprot(properties are defined later in this document.). When one is added to this value, it can be passed to FTGetProtocolName to retrieve the name of the protocol that can then be used to set the FT ActiveX object's protocol property.

str FTGetSendFileList()

This function returns the list of files that are to be sent for the current transfer. This is the contents of property 265 LastSendSrc (properties are defined later in this document.)

SET PROPERTIES

These functions allow the programmer to set basic properties relating to common file transfer situations. All the functions below will, when called, set not only the current file transfer parameters, but will set the properties saved in the .TPX file which includes those that show up on the file transfer dialogs.

void FTSetSync(bOn)

FTSetSync will turn on and off the synchronous transfer of files, meaning that when it is on, script will not execute until the current file transfer has completed. If you wish to turn on synchronous file transfers, you must do so before EACH transfer.

bOn - set to true to turn on synchronous file transfers, false to turn it off.

void FTSetAsciiMode(bOn)

FTSetAsciiMode will turn on and off ASCII CR/LF conversion.

bOn - set to true to turn on ASCII CR/LF conversion, false to run in binary mode.

void FTSetXferStat(bOn)

FTSetXferStat turns on and off the file transfer status screen during a transfer.

bOn - set to true to display the status screen, false to turn it off. This variable is set to true by default.

void FTSetPassword(sPassword)

FTSetPassword sets the password used for FTP file transfers. When this is called, it will automatically set FTP to use its own login and password, as opposed to being anonymous or using the emulator login and password.

sPassword - is the password to be set.

void FTSetUsername(sUser)

FTSetUsername sets the user name used for FTP file transfers. When this is called, it will automatically set FTP to use its own login and password, as opposed to being anonymous or using the emulator login and password.

sUser - is the user name to be set.

void FTSetHostname(sHost)

FTSetHostname sets the host name used for FTP file transfers. When this is called, it will automatically set FTP to use its own host name instead of the current emulation host name.

sUser - is the user name to be set.

void FTSetProtocol(sProtocol)

FTSetProtocol sets the protocol to be used for the next file transfer.

SProtocol - is a string value that is the protocol to be used for file transfers.

This must be one of the following values:

TERMCRC
WTERMCRC
FTP
XMODEM
XMODEM-CRC
YMODEM
ZMODEM
KERMIT
ASCII
LINE

void FTSetCompression(bCompress)

FTSetCompression turns on and off file transfer compression.

bCompress - is a boolean true or false value. TRUE for compression on, FALSE for compression off.

Function Reference

FILE TRANSFER INITIATION FUNCTIONS

These are functions that are used mainly to start an initial file transfer.

void FTSend(sFiles, sRemotedir, iDlgID)

FTSend will send one or more files to a remote system. The behavior of this command differs depending upon the arguments passed and the protocol being used. In the case of FTP file transfers, this function will create a new file transfer object and associate it with the current session. This is because FTP file transfers are not associated with an emulator session.

sFiles - names the files which are to be sent. These must be space separated. If a filename or its path contains a space, it should be enclosed within double quotes. For example, the file

c:\program files\Century\test.txt

needs to be enclosed within quotes

“c:\program files\Century\test.txt”

so that it is not treated as two files

(c:\program and files\Century\test.txt)

sRemotedir - names the remote directory in which received files are placed. This is optional, and is valid only for the following protocols:

WTERMCRC, FTP

Using these two protocols, the sending side will determine where to place received files. This overrides the receiver's local destination directory. If it is not specified, the current directory of the receiver is used. In all other cases, the remote host will determine where the files should be placed when received.

iDlgID - This value is reserved for internal functions and should always be set to 0.

void FTRecv(sFile,sDirectory)

FTRecv will passively receive one or more files from a remote sender. In the case of FTP file transfers, this function will eventually create a new file transfer object and associate it with the current session. This is because FTP file transfers are not associated with an emulator session.

sFile - names the file which is to be received. This is valid only for the following protocols:

XMODEM, XMODEM-CRC

In all other cases, the filename is passed along as part of the protocol. The filename specified in sFile is overridden by the sender's filename when using these protocols.

sDirectory - names the directory in which files that are received are placed. This is overridden by the remote side when using WTERMCRC.

void FTGet(sFileList,sDirectory)

FTGet will actively retrieve files from a remote host computer. In the case of FTP file transfers, this function will eventually create a new file transfer object and associate it with the current session. This is because FTP file transfers are not associated with an emulator session.

sFileList - names the files which you wish to receive. These must be space separated. If a filename or its path contains a space, it should be enclosed within double quotes. For example, the file

c:\program files\Century\test.txt

needs to be enclosed within quotes

"c:\program files\Century\test.txt"

so that it is not treated as two files

(c:\program and files\Century\test.txt)

sDirectory - names the directory into which the received files are placed.

FTGet works only with the following protocols:

TERMCRC, WTERMCRC, FTP

When FTGet is used with any of the other protocols (i.e., XMODEM), the result is the same as if FTRecv was used.

When using FTGet with the FTP protocol, multiple file receives are not possible when the files are explicitly listed. For example, passing the following string as the sFileList parameter will result in a “Cannot Read Remote Directory” error: “ftpctest/connect.gif ftpctest/testfile.txt”. However, wildcards can be used, as in “ftpctest/*.gif” or “ftpctest/*.txt”, but cannot be combined as in “ftpctest/*.txt ftpctest/*.gif” passed as one argument.

FILE TRANSFER UTILITY FUNCTIONS

These functions are a collection of miscellaneous utilities that perform common tasks necessary when setting up a file transfer operation.

str FTGetProtocolName(nProtocolIndex)

FTGetProtocolName will return the name of the protocol (as a string) associated with the index passed in. This is useful when setting the FT ActiveX object’s protocol property which takes a name rather than an index.

nProtocolIndex - The 1-based index of the protocol name you wish to retrieve. The value stored in the .TPX file is zero-based, meaning that you must add one to it before calling this function.

object FindSessionFromFTObj(ft)

This function returns the zero-based index of the FT session associated with the passed in FT ActiveX object. It returns -1 if it cannot find a session. See also the public variable section for a description of the variables that make up the FT sessions.

ft - The ft ActiveX object that you wish to find the associated FT session for.

object FindFTFromStatus(oDlg)

oDlg -The status dialog object that you wish to find the associated FT object for.

This function returns the ActiveX FT object associated with a given file transfer status dialog object. It returns nil if it cannot find an associated FT object.

object FindSessionByFT(oFT)

oFT - The ft ActiveX object that you wish to find the associated TE session for.

Events Reference

FILE TRANSFER CALLBACK FUNCTIONS

These functions are designed to be overridden by a CSL programmer to insert custom code into the chain of events that happen when a file transfer occurs.

void FTStartSend(oFT, nSess)

FTStartSend is a callback function that is triggered when a file transfer SEND or XFER is initiated. More precisely, it occurs as the first when the internal callback function `ft_StartBatchSend` is called, which itself is fired by the FT ActiveX control when a SEND or XFER operation is performed.

oFT - is the actual FT ActiveX object that is performing the transfer in question. This can be used to get information from the control, check error status or even change parameters (though this is not recommended, and doesn't work with all settings.)

nSess - is the numeric index of the TT session that is currently associated with this transfer. This can be used to check properties of the session, display information to the TE control, change sessions or any number of other things. This parameter will be set to -1 in the case of FTP file transfers, as these are not directly associated with a single session.

void FTEndSend(oFT, nSess)

FTEndSend is a callback function that is triggered when a file transfer SEND or XFER has finished. Specifically, it is called first in the internal callback function `ft_EndBatchSend`, which is fired by the FT ActiveX control when a SEND or XFER operation has completed.

oFT - is the actual FT ActiveX object that is performing the transfer in question. This can be used to get information from the control, check error status or even change parameters (though this is not recommended, and doesn't work with all settings.)

nSess - is the numeric index of the TT session that is currently associated with this transfer. This can be used to check properties of the session, display information to the TE control, change sessions or any number of other things. This parameter will be set to -1 in the case of FTP file transfers, as these are not directly associated with a single session.

void FTStartReceive(oFT, nSess)

FTStartReceive is a callback function that is triggered when a file transfer GET or RECV has been started. Specifically, it is called first in the internal callback function `ft_StartBatchReceive`, which is fired by the FT ActiveX control when a GET or RECV operation has been initiated.

oFT - is the actual FT ActiveX object that is performing the transfer in question. This can be used to get information from the control, check error status or even change parameters (though this is not recommended, and doesn't work with all settings.)

nSess - is the numeric index of the TT session that is currently associated with this transfer. This can be used to check properties of the session, display information to the TE control, change sessions or any number of other things. This parameter will be set to -1 in the case of FTP file transfers, as these are not directly associated with a single session.

void FTEndReceive(oFT, nSess)

FTEndReceive is a callback function that is triggered when a file transfer GET or RECV has finished. Specifically, it is called first in the internal callback function `ft_EndBatchReceive`, which is fired by the FT ActiveX control when a GET or RECV operation has completed.

oFT - is the actual FT ActiveX object that is performing the transfer in question. This can be used to get information from the control, check error status or even change parameters (though this is not recommended, and doesn't work with all settings.)

nSess - is the numeric index of the TT session that is currently associated with this transfer. This can be used to check properties of the session, display information to the TE control, change sessions or any number of other things. This parameter will be set to -1 in the case of FTP file transfers, as these are not directly associated with a single session.

Sample File Transfer Script

The following is a simplified file transfer script using the FTP protocol. The script connects to the host using the username and password specified. It then sends and receives a file.

```
FTSetProtocol("FTP");
//Sets the file transfer protocol

FTSetHostName("255.255.255.255");
//Sets the host name that you would like to connect to

FTSetUsername("testuser");
//Sets the user name to be //used for login

FTSetPassword("userpassword");
//Sets the password to be used for login

FTSetSync(1);
//Pauses script execution until after the transfer is finished

FTSend("c:\\autoexec.bat", ".", 0);
//Sends the autoexec.bat file to the current directory on the host

FTGet("/usr/testuser/test.fil", ".");
//Gets the test.fil file from the host and places it in
//the current working directory on the PC
```

Chapter 5

TERM Script to CScript Translation Guide

In this chapter

Overview

Translating TERM Script Language to CScript

TSL/CSL Translation Reference

Beginning with the 4.05 version of the TinyTERM emulator, Century Software has added backward compatibility with previous versions of the TERM emulator's scripting language. Although the current version still does not provide full backward compatibility, you will find that the most commonly used TERM Script language commands now work in the TinyTERM emulator product.

Translating TERM Script Language to CScript

RUNNING TERM SCRIPTS WITH TINYTERM

To run TERM Script language scripts, the TinyTERM emulator first translates the TERM Script language script to CScript. See section 3, "TERM Script to CScript Translation Table," for a complete list of CScript commands and functions generated by the TERM Script translator. If the translator can translate the TERM Script commands without any errors, then the TinyTERM emulator automatically runs the translated script.

EXECUTING TERM SCRIPT LANGUAGE

You can execute TERM Script language scripts from within the Script Editor dialog box or with the Execute Script File command on the Tools menu or the ribbon bar.

1. Choose Script Editor from the Tools menu.
 2. Click Open, and choose the TERM Script language script (usually a .cmd file) you want to execute.
 3. Click Run.
- or*
1. Click the Execute Script button on the ribbon bar, or choose Execute Script File from the Tools menu.
 2. From the dialog box, choose the script you want to execute and click OK.

You can also set session properties in the TinyTERM emulator's user interface to execute TERM Script language scripts post-startup, post-session start, post-connect, post-login, pre-logout, pre-disconnect, pre-session close, and pre-application exit.

TRANSLATION ERRORS

Supported TSL commands will translate and run without modification. TSL scripts that do not execute will generally have one of these problems:

- Uses commands not supported in the current version of the TinyTERM emulator
- Contains syntax errors
- Uses macro expansion that the translator cannot handle

UNSUPPORTED TERM SCRIPT LANGUAGE FEATURES

This version of the TinyTERM emulation does not support the following TSL features:

- Dynamic macro expansion. Where possible, the translator will make the proper substitutions for macro expansion, but in many cases, the translator will not be able to determine how a macro will be used during runtime and cannot translate TSL macros
- Any command not displayed in bold in the translation table (see section 3)

TRANSLATION TIPS

SCRIPT DEBUGGING

The TSL to CScript translator was designed to translate working TSL scripts, not for debugging new scripts. However, in the event that problems do arise, there are tools available to help localize them. While translation errors can be easily diagnosed through translator error messages (see Century TSL Script Translator documentation), runtime errors are another issue. For this, TinyTERM has a debug mode which can be started from the command line.

1. From the TinyTERM install directory, type “dbmon32\dbmon32”. This will start the Century Debug Monitor.
2. Again from the TinyTERM install directory, type “tt -debug”. This will start the TinyTERM emulator in debug mode.

In debug mode, TinyTERM will display messages regarding its current state and script command execution to the debug monitor.

MACRO EXPANSION

All macro expansions not contained within quotes must be followed by a digit 0-9 or the ‘(‘ character.

\$1	OK
\$2	OK
\$(expr)	OK
\$(toupper(g))	OK
file\$(opts)	OK
x\$(y)z	OK
“\$expr”	OK
\$expr	ERROR

Macro expanded keywords cannot have spaces within the parentheses.

send c:\data.txt	OK
send \$(filename)	OK
send \$(filename)	ERROR

Many command parameters which expect unquoted strings can be macro expanded. However, some items (such as keywords for some commands and variable names) cannot be macro expanded. In such instances, it may be necessary to enumerate all possibilities in a switch block.

capture disk \$(file) overwrite	OK
capture disk \$(file)\$ (opts)	ERROR

As a workaround, use something like this:

```
switch
case opts="overwrite"
    capture disk $(file) overwrite
case opts="append"
    capture disp $(file) append
endswitch
```

There are some instances where a macro expansion in a non-keyword parameter will not translate properly. This is the case with the first parameter in the setkey command:

```
setkey f1 SCRLINEUP          OK
```

correctly translates to:

```
tsl _setkey _cmd( 256, 14 /*SCRLINEUP*/);
```

```
setkey $(x) SCRLINEUP      ERROR
```

incorrectly translates to:

```
tsl _setkey _cmd( _str(x), 14 /*SCRLINEUP*/);
```

The best workaround for this is to edit the translated code:

```
tsl _setkey _cmd( tsf _keyid(x), 14 /*SCRLINEUP*/);
```

Procedure calls do not allow macros to expand to procedure names. The only workaround for this is to enumerate all possibilities in a switch or if-then block.

```
do @$(action _proc)      ERROR
```

As a workaround, use something like this:

```
switch
case action _proc="get _files"
    do @get _files
case action _proc="calc _values"
    do @calc _values
endswitch
```

Macro expansions are not allowed within array index brackets. Fortunately, integer variables may be used directly within array index brackets, which completely solves the problem.

```
display strList[index]    OK
```

```
display strList[$(index)] ERROR
```

Macro expanded indexed variables are not correctly translated. An easy workaround for this is to first assign the indexed variable to another, non-indexed variable, and then to macro expand that.

```
send $(fileList[2])      ERROR
```

As a workaround, use something like this:

```
setvar tempFile fileList[2]
```

```
send $(tempFile)
```

There may be instances where it is desirable to place something that looks like a macro expansion (but does not actually expand) within a string. The macro expansion dollar sign ('\$') can be escaped (using the backslash character '\'), with a restriction: text may not immediately follow it. In order to make text immediately follow an escaped dollar sign without confusing the translator, a separate string must be used.

```
display "This looks like a \$macro expansion"      ERROR
```

As a workaround, use something like this:

```
display "This looks like a \$" + "macro expansion"
```

PROCEDURE CALLS

Inter-file procedure calls do not work quite the same way as they did in TinyTERM 3.3. To make any inter-file procedure call, the target file must first be manually loaded into memory (the script must be executed). Calls to these procedures must contain the file name, but cannot contain the path or directory.

```
do fileops@getfiles          OK
do c:\scripts\fileopts@getfiles  ERROR
```

Procedure call parameters are subject to some limitations. Specifically, concatenating strings in-line with a procedure call will not translate properly.

```
do @getFiles path+"\\"+filename  ERROR
```

incorrectly translates to:

```
tsl _do(scriptname _getFiles("path+"\\"+filename));
```

As a workaround, use something like this:

```
setvar fullPath pat+"\\"+filename
do @getFiles fullPath
```

NAMING ISSUES

There are some important issues with naming which arise as translation side effects and as differences between TERM Script and CScript. One such issue relates to script filenames, and the other relates to the difference between TSL and CSL reserved words.

The first problem exists because TERM Script “.CMD” files translate such that all global data and operations are contained within one “global” CScript function. This “global” CScript function is named after the TERM Script file. This does not usually cause problems, but if the file happens to have the same name as a translated TSL command, this “global” CScript function will override the corresponding TSL/CScript function. For example, a if a file called “`tsl_msgbox.cmd`” were to contain the following line:

```
msgbox "Hello!"
```

It would translate to the following:

```
function
tsl _msgbox()
{
    tsl _msgbox( "Hello!", "", 0);
    return( tsl _return( 0));
}
```

This new global function “`tsl_msgbox`” overrides the original `tsl_msgbox` command, which in this particular case will cause an infinite loop.

The second problem exists because certain variable names which are perfectly valid in TERM Script Language are reserved words in CScript. For example, “`var`” might be used as a variable name in TSL. In CScript, `var` is a keyword which can be used to declare variables. This works properly in TSL:

```
setvar var 5
```

Whereas what it translates to will not run properly:

```
var = 5;
```

Command Line TSL Script Translator

This information describes the TSL Script Translator included in TinyTERM. It will help you use the translator to convert TSL scripts to CScript. For more information on the Term Script Language, see the TSL Command Reference.

INSTALLATION

TSLTRANS.EXE, the TSL Script Translator, is installed with TinyTERM Plus Edition, TinyTERM Web Server Edition and TinyTERM Thin Client Edition. It can be found in the installation directory, normally C:\Program Files\Century\TinyTERM. No additional installation or configuration is necessary.

USAGE

The TSL Script Translator (TSLTRANS.EXE) will convert scripts from the TERM Script Language (TSL) included in TinyTERM 3.3 to the CScript language included with the current version of TinyTERM. The command syntax is:

```
tsltrans.exe [-n] <scriptname>.cmd
```

Replace <scriptname> with the name of your TSL Script. The optional -n parameter will cause TSLTRANS.EXE to halt when it encounters an error, until the Enter key is pressed.

A CScript file named <scriptname>.cs will be created. In many cases this file will run properly with no modification; however, The TSL Script Translator is not a turnkey solution. It is intended as a preprocessor for script translation only. Many translated scripts will need to be edited before they will function as originally intended. Refer to the Known Problems for more information.

The translator works by having some built-in knowledge of TSL commands and functions. Each TSL command is translated to a CScript function by prepending “tsl_” to the command name, and re-writing the syntax to be CScript compatible. TSL built-in functions are treated the same way, but “tsf_” is prepended instead. The TSL outer loop is converted into a CScript function with the same name as the .cmd filename, and TSL procs are converted to filename_procname format. Because true is represented as -1 in TSL, and 1 in CScript, boolean expressions are converted in a more complex fashion.

We have created a TSL Compatibility Library, written in CScript, that attempts to emulate the TSL commands completely. Keep in mind that you can always write your own tsl_ and tsf_ CScript functions to completely control the new script.

KNOWN PROBLEMS

UNIMPLEMENTED COMMANDS

Some TSL commands have no CScript equivalents. We have not currently implemented any of the user-interface TSL commands in CScript. This is because of the major design differences between the older and newer implementations of the user interface. These commands will translate, but their implementation produces an error message when run. See the TSL Translation Guide for a list of which commands are not implemented.

MACRO EXPANSION

CScript does not support macro expansion. Some TSL scripts that use macro expansion will cause TSL-TRANS.EXE to generate an error on each such line. In particular, it is only possible to translate macro expansion that occurs within a quoted literal string, or where an unquoted string is expected. Macro expansion outside this case is impossible to translate accurately, without actually running the script.

Script translation errors are displayed on standard output, and the resulting CScript file retains the non-translated line commented out. Most lines can be converted to use variables instead of macro expansion. For the example, the following can be used to manually translate the following line:

```
setvar err $_errno  
err = _errno;
```

ERROR #442 (NO STRINGS FILE)

You will see the message “Error #442 (no strings file)” if the file censtr32.dll is not in the same directory as tsltrans.exe. If you move the tsltrans.exe to another location than its installed directory, ensure that you also copy the censtr32.dll file to the same directory where you moved tsltrans.exe.

UNINSTALLING

TinyTERM can be uninstalled by going to Add/Remove Programs in the Control Panel. Other methods (deleting directories or files, for example) can cause system instabilities or problems with future installation of this product. The uninstall is based on a log file created during installation. The uninstall will not remove any files currently in use or files manually moved or altered after the original installation.

TSL/CSL Translation Reference

To run TERM Script language commands, the TinyTERM emulator must first translate the TERM Script language commands into CScript commands. Most of these commands are part of a compatibility library. (In fact, a future release of CScript might require that you load a TSL compatibility library before using these commands.)

The following lists show all the TSL commands and functions that the translator will correctly handle. The TinyTERM emulator, version 4.05, only supports those commands and functions listed in bold text.

CONFIGURATION SETTINGS

CAPTURE keyword filename tsl_capture(str, file)

keyword = none, disk, device, prtmgr, file, spool, terminal, on, off, close

filename may be immediately appended by (a), (o), (f), (b), or (m)

CDELAY ilit tsl_cdelay(n)

COLOR keyword fcol, bcol tsl_color(str, str-fg, str-bg)

keyword = default, reverse, blink, underline, bold, dim

fcol = default, black, blue, green, cyan, red, magenta, brown, white

bcol = default, black, blue, green, cyan, red, magenta, brown, white, ltgrey, ltblue, ltgreen, ltcyan, ltred, ltmagenta, yellow, ltwhite

COLOR NUMBER ivar tsl_color_number(n)

COMBASE ivar, ivar, ivar tsl_combase(n, n, n)

DANSOFF svar tsl_dansoff(str)

DAUTOANS svar tsl_dautoans(str)

DCONNECT svar tsl_dconnect(str)

DESCAPE svar tsl_descape(str)

DHANGUP svar tsl_dhangup(str)

DINIT svar tsl_dinit(str)

DOFFHOOK svar tsl_doffhook(str)

DOKSTR svar tsl_dokstr(str)

DPREFIX svar tsl_dprefix(str)

DRINGSTR svar tsl_dringstr(str)

DSUFFIX svar tsl_dsuffix(str)

DTIMEOUT ilit tsl_dtimeout(n)

DDE ENABLE tsl_dde_enable()

DDE DISABLE tsl_dde_disable()

DDE NAME svar tsl_dde_name(str)

DDE TIMEOUT ivar tsl_dde_timeout(n)

DEVPREFIX svar tsl_devprefix(str)

EDITOR filename tsl_editor(file)

EMAILADDR svar tsl_emailaddr(str)

EMAILsubj svar tsl_emailsubj(str)

EXTGEN svar	tsl_extgen(str)
EXTXFER svar	tsl_extxfer(str)
FILLCHAR chlit	tsl_fillchar(n)
FLAGS svar	tsl_flags(str)
FONT svar [, ivar [, ivar]]	tsl_font(str, nh, nw)
FTPHOST svar, svar, svar	tsl_ftphost(strhost, struser, strpass)
HIWAIT ilit	tsl_hiwait(n)
HTIME ilit	tsl_hptime(n)
ITIME ilit	tsl_itime(n)
JUMPSROLL ilit	tsl_jumpscroll(n)
KEYBOARD keyword	tsl_keyboard(str)
keyword = bios, extbios, int9, int9n	
LCHAR	tsl_lchar(n)
LDELAY ilit	tsl_ldelay(n)
LOCKTIME ilit	tsl_locktime(n)
LOCKFILE filename	tsl_lockfile(file)
LOGFILE filename[(o)]	tsl_logfile(file)
LOGFILE CLOSE	tsl_logfile_close(n)
MIMEENCODE filename, filename	tsl_memeencode(str, str)
MIMEDECODE dvar, filename	tsl_memedecode(str, str)
MODEM filename	tsl_modem(file)
MODEM HAYES	tsl_modem("hayes")
NETDIALOG svar	tsl_netdialog(str)
NETNAME svar	tsl_netname(str)
NETPORT ilit	tsl_netport(n)
NETSTIME ilit	tsl_netstime(n)
NETTERM svar	tsl_netterm(str)
NETVTNAME svar	tsl_netvtname(str)
NETWORK svar	tsl_network(str)
PAGES ivar	tsl_pages(n)
PRINTER keyword [filename]	tsl_printer(str, file)
keyword = none, disk, device, prtMgr, spool, terminal, on, off, capture, close	
PROTOCOL keyword	tsl_protocol(str)
keyword = xon, rts, etx, none	
PROTOCOL QUIET keyword	tsl_protocol_quiet(str)
keyword = on, off	
PROTOCOL DCD keyword	tsl_protocol_dcd(str)
keyword = on, off	
REDIAL ilit	tsl_redial(n)
RESTART ilit	tsl_restart(n)
RETRIES ilit	tsl_retries(n)
RTIME ilit	tsl_rtime(n)

SCROLLBACK ivar	tsl_scrollback(str)
SET ABORT keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ADDCR keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ADDEOF keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ADDLF keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ALTCHK keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ALTKEYS keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET BLOCKMODE keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET COMPRESS keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET CONSXON keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET CONTROL keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET DDTR keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET DESTBS keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET DIMDIM keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET DLGUNITS keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET DTR keyword keyword = on, off, 1, 0	tsl_set(int, str)

SET ECHO keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET EMAILDEL keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET EOFOPEN keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ERROR keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ERRORBOX keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ESC8BIT keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET ESCCTL keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET EXITDISC keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET EXITDTR keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET EXPLODE keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET EXTEND keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET FILECONV keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET KERMECHO keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET LOG keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET MASKPAR keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET NOSCROLL keyword keyword = on, off, 1, 0	tsl_set(int, str)

SET REVDIM keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET RTS keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET SCALING keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET SOUND keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET TABEX keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET TOUPPER keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET VIEW keyword keyword = on, off, 1, 0	tsl_set(int, str)
SET XFERSTAT keyword keyword = on, off, 1, 0	tsl_set(int, str)
SETKEY keyname keyword keyname = menu, help, gold, shell, capture, print, compose, break, abort, scrlineup, sclinedn, scrpageup, scrpagedn, scrright, sclleft, scanmode, nextsess, setup, exit, hotkey	tsl_setkey_cmd(n, str)
SETKEY keyname str	tsl_setkey_macro(n, str)
SETKEY RESET ivar	tsl_setkey_reset(n)
STIME ilit	tsl_stime(n)
TERMID svar	tsl_termid(str)
TDELAY ilit	tsl_tdelay(n)
TOOLTIPS ivar	tsl_tooltip(n)
TRANS keyword [svar, svar] keyword = input, output, display	tsl_trans(str, str, str)
TRANS RESET	tsl_trans_reset()
VIDBLINK keyword keyword = nobios, bios	tsl_vidblink(str)
VIDCARD CUSTOM, ax, bx	tsl_vidcard_custom(nax, nbx)
VIDCARD keyword keyword = none, graphics, ahead, ati1, ati2, everex, paradise, tecmar, tsent, video7	tsl_vidcard(str)

VIDOUT keyword keyword = ram, snow, int10, bios, graphics	tsl_vidout(str)
VIDUNDER keyword keyword = none, font, port	tsl_vidunder(str)
WILDSIZE ilit	tsl_wildsize(n)
WRUCHAR chlit	tsl_wruchar(n)
XONXOFF ivar, ivar	tsl_xonxoff(n, n)

CURRENT CONNECTION CONFIGURATION SETTINGS

AUTOCMD svar	tsl_autocmd(str)
BAUD word	tsl_baud(str)
CHARSET keyword keyword = american, british, dutch, finnish, french, canadian, german, italian, danish, norwegian, portuguese, swedish, swiss	tsl_charset(str)
EMULATE keyword keyword = vt320, vt320-7, vt220, vt220-7, vt100, vt102, vt52, ansi, at385, scoansi, wyse60, wyse60-25, wyse-50, tv950, tv925, tv912, adm1, imb3101, tty	tsl_emulate(str)
LOGIN	tsl_do_login()
LOGIN svar	tsl_login(str)
LOGOUT	tsl_do_logout()
LOGOUT svar	tsl_logout(str)
MODE keyword keyword = full, half, dump, mnemonic, control	tsl_mode(str)
NODE svar	tsl_node(str)
PARITY keyword keyword = none, odd, even, space, mark	tsl_parity(str)
PASSWORD svar	tsl_password(str)
PORT svar	tsl_port(str)
REMARK eolstring	tsl_remark(str)
SET BSDEL keyword keyword = on, off, 1, 0	tsl_set_bsdel(str)
SET WRAP keyword keyword = on, off, 1, 0	tsl_set_wrap(str)

STARTSERVER	tsl_do_startserver()
STARTSERVER svar	tsl_startserver(str)
STOPSERVER	tsl_do_stopserver()
STOPSERVER svar	tsl_stopserver(str)
STOPBITS ilit	tsl_stopbits(n)
USER svar	tsl_user(str)
WORDLEN ilit	tsl_wordlen(n)
WRU svar	tsl_wru(str)
XPROT keyword	tsl_xprot(str)
keyword = termcrc, wtermcrc, xmodemcrc, kermit, compsrvb, xmodem, ymodem, zmodem, ftp, ascii, line, external	

USER INTERFACE COMMANDS

AT ivar, ivar	tsl_at_clreol(ny, nx)
AT ivar, ivar svar	tsl_atobjinit()
	tsl_atobjdone(nid, ntype, nx, ny, nw, nh, str, nflags, nint1, nint2, echoch, varstr)

AT ivar,ivar [ID #] [TABSTOP] BUTTON svar, retval, [keyname]
 tsl_atobjinit()
 tsl_atobjdone(nid, ntype, nx, ny, nw, nh,
 str, nflags, nint1, nint2, echoch, varstr)

AT ivar,ivar [ID #] [TABSTOP] keyword [ivar[,ivar[,ivar[,svar]]]]
 tsl_atobjinit()
 tsl_atobjdone(nid, ntype, nx, ny, nw, nh,
 str, nflags, nint1, nint2, echoch,varstr)
 keyword = text, box, icon, picture, meter, edit, checkbox, radio, keyin

AT ivar,ivar [ID #][TABSTOP] GET var keyword [ivar[,ivar[,ivar[,svar]]]]
 LIST svar [,svar]...
 tsl_atobjinit()
 tsl_atobjaddlist(svar)
 tsl_atobjaddlist(svar)...
 tsl_atobjdone(nid, ntype, nx, ny, nw, nh,
 str, nflags, nint1, nint2, echoch, varstr)

AT ivar,ivar [ID #] [TABSTOP] GET var keyword [ivar[,ivar[,ivar[,svar]]]]
 ARRAY svar
 tsl_atobjinit()
 tsl_atobjaddarray(var)
 tsl_atobjdone(nid, ntype, nx, ny, nw, nh,
 str, nflags, nint1, nint2, echoch, varstr)
 keyword = popup, elist, listbox

```

AT ivar,ivar [ID #] [TABSTOP] GET var
    CLASS keyword, [ivar[ ,ivar[ ,ivar[ ,svar]]]]
    tsl_atobjinit();
    tsl_atobjdone( nid, ntype, nx, ny, nw, nh,
        str, nflags, nint1, nint2, echoch, varstr)
keyword = cenEditBox, cenIconList, cenIcon, cenPicture, cenFontList

AT CLEAR                                tsl_at_clear( );

AT READ [SAVE keyname] [CALLBACK procstring] [NOWAIT]
    tsl_at_read( nflags, str, key )

AT SAVE ivar                            tsl_at_save( n )
ATTR keyword                            tsl_attr( str )
keyword = default, reverse, blink, underline, bold, dim

BELL                                    tsl_bell( )
CLS                                    tsl_cls( )
CURSOR keyword                          tsl_cursor( str )
keyword = wait, arrow

DISPLAY [expr [,expr]...]              tsl_display( expr1 )
INIDELSTR svar, svar                   tsl_inidelstr( str, str )
INIENUMSECT dvar, filename, svar       tsl_inienumsect( str, file, str )
INIGETFF dvar, filename, svar, ivar    tsl_inigetstr( str, file, str, n )
INISSETFF dvar, filename, svar, ivar   tsl_inisetstr( str, file, str, n )
INISSETFILE filename                  tsl_inisetfile( str )
INISSETSTR svar, svar, svar            tsl_inisetstr( str, str, str )
INISYNC filename, ivar                tsl_inisync( file, n )
MSGBOX svar[,svar[,ivar]]             tsl_msgbox( str1, str2, n )

MENU ivar OPEN svar [ivar, ivar, [ivar, ivar] ,ivar]
    [PARENT ivar] [HINT svar]
    tsl_menu_open( nid, npid, strtitle,
        strhint, ny, nx, nfg, nbg, flags )

MENU ivar ENTRY svar [EXECUTE svar] [ID ivar]
    [HINT svar] [svar [EXECUTE...]]..
    tsl_menu_entry( pid, itemid, stentry1,
        strhint, strex )

MENU ivar SHOW keyword                 tsl_menu_show( nid, str )
keyword = menubar, lotus, popup

MENU ivar STATUS ivar, ivar           tsl_menu_status( nid, nitemid, nflags )
MENU ivar keyword                      tsl_menu_close( str )
keyword = close, off, restore

```

OBJSETFONT ivar, ivar, svar, ivar [,ivar] tsl_objsetfont(n, n, str, n, n)	
OBJPAINT ivar [,ivar] OBJADDLIST ivar, ivar, svar PASTE keyword keyword = link, text	tsl_objpaint(n, n) tsl_objaddlist(n, n, str) tsl_paste(str)
PAUSE eolstring QUIT ilit REDRAW SCREEN keyword keyword = copy, print, noselect	tsl_pause(str) tsl_quit(n) tsl_redraw() tsl_screen(str)
SETCAPTURE ivar, ivar STDFILE var [,svar [,filename [,svar [,svar [,ivar]]]]] var = tsl_stdfile(str, file, str, str, n)	tsl_setcapture(n, n)
STDSAVE var [,svar [,filename [,svar [,svar]]]] var = tsl_stdsave(str, file, str, str, n)	
WCLOSE ivar [,ivar]... WHIDE ivar [,ivar]... WMOVE ivar, ivar, ivar[, ivar] WOPEN ivar, ivar, ivar, ivar, ivar[, ivar, ivar[, ivar[, svar[, svar]]] tsl_wopen(n, ny,nx, nh,nw, nfg,nbg, nstyle, strttitle, strlable)	tsl_wclose(n1) tsl_whyde(n1) tsl_wmove(n, ny, nx, n) tsl_wopen(n, ny,nx, nh,nw, nfg,nbg, nstyle, strttitle, strlable)
DLGOPEN ivar, ivar, ivar, ivar, ivar[, ivar, ivar[, ivar[, svar[, svar]]] tsl_dlgopen(n, ny,nx, nh,nw, nfg,nbg, nstyle, strttitle, strlable)	
WSELECT ivar WSHOW ivar[,ivar]... WTITLE ivar, svar	tsl_wselect(n) tsl_wshow(n1) tsl_wtitle(n, str)

FILE TRANSFER COMMANDS

GET filelist filename filelist may be appended by (q) or (x) filename may be appended by (c), (f), (t), or (z)	tsl_get(str)
RCV filename filename may be appended by (c), (i), (s), (t), or (z)	tsl_rcv(str)
REMOTE eolstring Note: only works with the QUIT command.	tsl_remote(str)

SEND filelist filelist may be appended by (c), (i), (s), (t), or (z)	tsl_send(str)
SERVER	tsl_server()
TIME filelist	tsl_time(file)
XFER filelist filename filelist may be appended by (q) or (x) filename may be appended by (c), (f), (t), or (z)	tsl_xfer(str)

GENERAL COMMANDS

ABORT [ilit]	tsl_abort(n)
ADDPMENTRY svar, svar, svar, svar, svar, svar tsl_addpmentry(str, str, str, str, str, n)	
ANSWER keyword keyword = off, now, wait, auto	tsl_answer(str)
BATCH [filename]	< no translation >
BREAK	tsl_break()
CONFIG GET [ivar]	tsl_config_get(n)
CONFIG NEW	tsl_config_new()
CONFIG SET	tsl_config_set()
CONFIG READ filename	tsl_config_read(file)
CONFIG WRITE filename	tsl_config_write(file)
CONFIG SYSREAD filename	tsl_config_sysread(file)
CONFIG SYSWRITE filename	tsl_config_syswrite(file)
CONNECT [filename]	tsl_connect(file)
COPY filename filename	tsl_copy(file,file)
CREAD var [,ivar [,ivar [,ivar]]]	var = tsl_cread(ntim, nchrs, n)
CREADINT var [,ivar [,ivar [,ivar]]]	var = tsl_creadint(ntim, nchrs, n)
DDE ADVISE ivar, svar, var	var = tsl_dde_advise(n, str)
DDE CLOSE ivar	tsl_dde_close(n)
DDE EXECUTE ivar, svar	tsl_dde_execute(n, str)
DDE INIT ivar, svar, svar	tsl_dde_init(n, str, str)
DDE POKE ivar, svar, svar	tsl_dde_poke(n, str, str)
DDE REQUEST ivar, svar, var	var = tsl_dde_request(n, str)
DDE UNADVISE ivar, svar	tsl_dde_unadvise(n, str)
DIALOG svar	tsl_dialog(str)
DWAIT keyword [,ivar]	tsl_dwait_carrier(str,n)
keyword = carrier, nocarrier	
DWAIT svar [, ivar]	tsl_dwait(str, n)
DWAIT ivar [, ivar]	tsl_dwait(n, n)
EXMIT svar	tsl_exmit(str)

FLUSH	tsl_flush()
GETPMLIST dvar	tsl_getpmlist(dvar)
HANGUP	tsl_hangup()
HELP [word]	tsl_help(str)
IGNORE keyword [svar]	[keyword [svar]]...
	tsl_ignore(str, str)
	keyword = input, output
IGNORE RESET	tsl_ignore_reset()
INTER	tsl_inter()
LEARN START filename	tsl_learn_start(file)
LEARN STOP	tsl_learn_stop()
LEARN PAUSE	tsl_learn_pause()
LEARN CONTINUE	tsl_learn_continue()
MAPKEY ivar, svar	tsl_mapkey(n, str)
READ [svar,] var	var = tsl_read(str)
READINT [svar,] var	var = tsl_readint(str)
READN [svar,] var	var = tsl_readn(str)
SESSION GOTO ivar [,ivar]	tsl_session_goto(n, n)
SESSION NEW [ivar]	tsl_session_new(n)
SESSION NEXT [ivar]	tsl_session_next(n)
SESSION DISCONNECT	tsl_session_disconnect()
SETCAP svar, svar	tsl_setcap(str, str)
SETCOLOR ivar, ivar	tsl_setcolor(n, n)
SETDELAY ilit tsl_setdelay(n)	
SETSYM ivar, svar [,ivar]	tsl_setsym(n, n, n)
TERMINAL NOCARRIER	tsl_terminal_nocarrier()
TERMINAL [svar][, keyname]	tsl_terminal(svar, n)
TYPE filename	tsl_type(file)
filename may be appended by (b)	
VERSION	tsl_version()
WAIT keyword [,ivar]	tsl_wait_carrier(str,n)
keyword = carrier, nocarrier	
WAIT svar [, ivar]	tsl_wait(str, n)
WAIT ivar [, ivar]	tsl_wait(n, n)
XMIT svar	tsl_xmit(str)

OPERATING SYSTEM LEVEL COMMANDS

CD filename	tsl_cd(file)
CHOWN slit filename	tsl_chown(str, file)
CHTYPE slit filename	tsl_chtype(str, file)
ERASE filename	tsl_erase(file)
EDIT eolstring	tsl_edit(file)
MKDIR filename	tsl_mkdir(file)
RENAME filename filename	tsl_rename(file, file)
RMDIR filename	tsl_rmdir(file)
RUN eolstring	tsl_run(str, 0)
RUNX eolstring	tsl_run(str, 1)

FILE I/O COMMANDS

FCLOSE ivar	tsl_fclose(n)
FCREATE ivar, svar	tsl_fcreate(n, str)
FFLUSH ivar	tsl_fflush(n)
FOPEN ivar, svar[, svar]	tsl_fopen(n, str, str)
FREAD ivar, var [,ivar]	tsl_fread(n, nchrs)
FREADLN ivar, var [,ivar]	tsl_freadln(n, nchrs)
FSEEK ivar, ivar[, ivar]	tsl_fseek(n, n, n)
FWRITE ivar, svar [,ivar]	tsl_fwrite(n, str, nchrs)
FWRITELN ivar, svar [,ivar]	tsl_fwriteln(n, str, nchrs)
LOG svar	tsl_log(str)

MISCELLANEOUS COMMANDS

DIMSTR var ivar	var = tsl_dimstr(n)
DIMINT var ivar	var = tsl_dimint(n)
DIMLOG var ivar	var = tsl_dimlog(n)
MEMLIST [word]	tsl_memlist(str)
RELEASE var [,var]...	tsl_release(var1)
RELEASE *	tsl_release("*")
SETVAR var expr	var = expr
SETVAR var _1	var = tsl_strarg(1)
SETVAR var \$1	var = tsl_macroarg(1)
SETVAR \$1 expr	tsl_setvar(tsl_strarg(1), expr)
SETVAR var _kermiteol	var = tsi_kermiteol()
SETVAR _kermiteol expr	tsl_setvar("_kermiteol", expr)
SETVAR _baud expr	ERROR
SETVAR _1 expr	ERROR

CONTROL AND FLOW COMMANDS

COMMENT eolstring	// str
! eolstring	// str
* eolstring	// str
DO procstring [arg] [arg]...	tsl_do(filename_proc(arg, arg,...))
EXECUTE svar	< no translation >
FOR svar IN (filelist) DO	tsl_forinit("filelist")
while ((svar = tsl_fornext()) != "") {	
ENDFOR	}
IF expr	if (expr) {
ELSEIF expr	} else if (expr) {
ELSE	} else {
ENDIF	}
IFERROR eolstring	tsl_noerror()
tsl_<command>	
if (tsi_errno()) {	
NOERROR eolstring	tsl_noerror(str)
tsl_<command>	
ON DELAY ilit	file_on_delay() {
ON CLOSE str	file_on_close(str) {
ON KEY keyname	file_on_key_n() {
ON error	file_on_error() {

ON abort	file_on_abort() {
ON carrier	file_on_carrier() {
ON nocarrier	file_on_nocarrier() {
ENDON	return(tsl_endon(3))
ONCLOSE procname	file_onclose(str)
ONDISCONNECT procname	file_ondisconnect(str)
PROC procname	file_procname() {
ENDPROC	return(tsl_return(0))
REPEAT	do {
UNTIL expr	} while (~expr)
RETURN [ilit]	return(tsl_return(n))
SWITCH	if (0) {
CASE expr	} else if (expr) {
DEFAULT	} else {
ENDSWITCH	}

PREVIOUS VERSION RETROFITTED COMMANDS

ADDCR keyword	tsl_addcr(str)
ADDLF keyword	tsl_addlf(str)
keyword = on, off	
BREAK keyword	tsl_break(str)
keyword = on, off	
CALL word	tsl_call(str)
INIT svar [,ivar]	tsl_init(str, n)
LCHRDY ilit	tsl_lchrdy(n)
LOWAIT ilit	tsl_lowait(n)
LSEND filename	tsl_send(str)
MINITEL ivar, ivar, ivar, ivar	tsl_minitel(n, n, n, n)
PARTOPEN svar	tsl_partopen(str)
SETKEY keyname keyword	tsl_setkey_cmd(n, str)
keyword = login, logout, cmd, newsess, gotosess, discsess	
SET REVMAP keyword	tsl_set_revmap(str)
SET SCROLLBAR keyword	tsl_set_scrollbar(str)
keyword = off, on	
TERMTYPE svar	tsl_termtype(str)
USEND filename	tsl_usend(str)
VI filename	tsl_edit(file)

TERM SCRIPT LANGUAGE FUNCTIONS

asc(svar)	tsf_asc(str)
atod(svar)	tsf_atod(str)
cdate(ivar)	tsf_cdate(n)
chname(ivar, ivar)	tsf_chname(n, n)
chr(ivar)	tsf_chr(n)
chrdy()	tsf_chrdy()
chrin()	tsf_chrin()
comin()	tsf_comin()
comst()	tsf_comst()
ctime(ivar)	tsf_ctime(n)
encrypt(svar)	tsf_encrypt(str)
eval(cmd)	tsf_eval(cmd)
exists(svar)	tsf_exists(str)
fconcat(svar, svar, svar)	tsf_fconcat(str, str, str)
feof(ivar)	tsf_feof(n)
fext(svar)	tsf_fext(str)
fhead(svar)	tsf_fhead(str)
field(svar, ivar, svar)	tsf_field(str, n, str)
filedate(svar)	tsf_filedate(str)
filemode(svar)	tsf_filemode(str)
filesize(svar)	tsf_filesize(str)
fname(svar)	tsf_fname(str)
ftell(ivar)	tsf_ftell(str)
getenv(svar)	tsf_getenv(str)
getexe(svar)	tsf_getexe(str)
gethomedir()	tsf_gethomedir()
getkey(int)	tsf_getkey(int)
getpath(svar, svar)	tsf_getpath(str, str)
getuserdir()	tsf_getuserdir()
hascolor()	tsf_hascolor()
inigetstr(svar, svar, svar)	tsf_inigetstr(str, str, str)
isconnect()	tsf_isconnect()
iscopy()	tsf_iscopy()
isdir(svar)	tsf_isdir(str)
ispaste(ivar)	tsf_ispaste(n)
keyid(svar)	tsf_keyid(str)
keyin()	tsf_keyin()
keyname(ivar)	tsf_keyname(n)
left(svar, ivar)	tsf_left(str, n)
len(svar)	tsf_len(tr)
mdmstat()	tsf_mdmstat()
menu(ivar, ivar)	tsf_menu(n, n)
menustat(ivar)	tsf_menustat(n)
midstr(svar, ivar, ivar)	tsf_midstr(str, n, n)
opsys()	tsf_opsys()

<code>opsys2(svar)</code>	<code>tsf_opsys2(str)</code>
<code>pos(svar, svar, ivar)</code>	<code>tsf_pos(str, str, n)</code>
<code>right(svar, ivar)</code>	<code>tsf_right(str, n)</code>
<code>str(ivar)</code>	<code>tsf_str(n)</code>
<code>strconv(svar, ivar)</code>	<code>tsf_strconv(str, n)</code>
<code>strhex(ivar, ivar)</code>	<code>tsf_strhex(n, n)</code>
<code>sum(svar, ivar, ivar)</code>	<code>tsf_sum(str, n, n)</code>
<code>sysvar(ivar)</code>	<code>tsf_sysvar(n)</code>
<code>termbits()</code>	<code>tsf_termbits()</code>
<code>termkey(ivar)</code>	<code>tsf_termkey(n)</code>
<code>time()</code>	<code>tsf_time()</code>
<code>tolower(svar)</code>	<code>tsf_tolower(str)</code>
<code>toupper(svar)</code>	<code>tsf_toupper(str)</code>
<code>trim(svar)</code>	<code>tsf_trim(str)</code>
<code>uniq(svar)</code>	<code>tsf_uniq(str)</code>
<code>val(svar)</code>	<code>tsf_val(str)</code>



CENTURY

S O F T W A R E

5284 South Commerce Dr. Suite C-134
Salt Lake City, Utah 84107

Toll Free (800) 877-3088
Phone (801) 268-3088
Fax (801) 268-2772
E-mail sales@centurysoftware.com
Web www.centurysoftware.com

SECURE CONNECTIVITY SOLUTIONS

Copyright © 1985-2005 Century Software, Inc. All Rights Reserved.
Century Software, TinyTERM Plus and the "hub" logo are trademarks or registered
trademarks of Century Software, Inc. All other trademarks, trade names, or company
names referenced herein are used for identification purposes only and are the
property of their respective owners.

CS-TTP-PRM-EN-032305