NAURTECH CORPORATION

CETerm Scripting Guide

for Version 5.1 or later

Naurtech Terminal Emulation and Web Browser Smart Clients

for Windows CE Devices

CETerm | CE3270 | CE5250 | CEVT220

Copyright Notice

This document may not be reproduced in full, in part or in any form, without prior written permission of Naurtech Corporation.

Naurtech Corporation makes no warranties with respect to the contents of this document and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Naurtech Corporation, reserves the right to revise this publication and referenced software without any obligation to notify any person or organization of such revision or changes.

Trademarks

CETerm®, CE3270™, CE5250™, CEVT220™ are trademarks of Naurtech Corporation.

Other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

Software Version

This document is for Version 5.1 or later of Naurtech Smart Clients.

Table of Contents

Copyright Notice 2
Trademarks 2
Software Version 2
Table of Contents 3
Preface 6
Assumptions 6
Conventions used in this Manual 6
Additional Documentation 7
Online Knowledgebase 7
1.0 Introduction 8
  1.1 Feature Highlights 9
2.0 Getting Started 11
  2.1 JavaScript Engine 11
  2.2 Enabling Scripting and Editing Scripts 12
    General Settings 12
    Editing Scripts 12
  2.3 CETerm Automation Objects 15
  2.4 IDA Action Codes 15
  2.5 OnBarcodeRead Script Event 15
  2.6 Automated Login 18
  2.7 Custom Screen Hot-Spots 20
3.0 CETerm Automation Objects 22
  3.1 The CETerm Object 22
    Methods 22
    Properties 26
  3.2 The Session Object 26
    Methods 27
    Properties 27
  3.3 The Screen Object 27
    Methods 27
    Properties 28
  3.4 The Browser Object 28
    Methods 28
    Properties 29
  3.5 The Message Object 29
    Methods 30
    Properties 30
  3.6 The TextInput Object 30
    Methods 30
    Properties 31
  3.7 The OS Object 31
    Methods 31
    Properties 33
  3.8 The File Object 34
    Methods 34
    Properties 36
  3.9 The Registry Object 37

Methods 37
Properties 39
4.0 CETerm Script Events 40
4.1 The OnBarcodeRead Event 40
Syntax 40
Example 41
4.2 The OnDocumentDone Event 41
Syntax 41
Example 42
4.3 The OnNavigateError Event 42
Syntax 42
Example For Windows CE 5.0 devices 43
Example For Windows Mobile devices 44
4.4 The OnNetCheckFailed Event 44
Syntax 44
Example 44
4.5 The OnSessionConnect Event 45
Syntax 45
Example 45
4.6 The OnSessionDisconnect Event 46
Syntax 46
Example 46
4.7 The OnSessionDisconnected Event 47
Syntax 47
Example 47
4.8 The OnSessionReceive Event 47
Syntax 47
Example 48
4.9 The OnSessionSwitch Event 48
Syntax 48
Example 48
4.10 The OnStylusDown Event 49
Syntax 49
Example 49
4.11 The OnWakeup Event 50
Syntax 50
Example 50
5.0 Scripting Techniques and Tips 52
5.1 Expect and ExpectMonitor for Automating Tasks 52
5.1.1 Expect Script 52
5.1.2 ExpectMonitor Class 53
5.1.3 Automating Tasks with Expect 57
5.2 Presenting Visual Feedback During Script Execution 57
5.3 Getting User Input to a Script 58
5.4 Running an External Program 59
5.5 Using Timers to Run Scripts 59
5.6 Accessing a File 60
5.7 Accessing the Registry 61
5.8 Writing Efficient Scripts 62
5.8.1 Use Local Variables 62
5.8.2 Encapsulate Code in Functions 63
5.8.3 Limit Execution Time 63

5.9 Debugging Scripts 64
5.9.1 Show Script Errors 64
5.9.2 OS.Alert() 65
Appendix 1 - IDA Action Codes 66
Appendix 2 - Properties 75
Application Properties 75
Device Properties 75
Session Properties 76
Scanner Properties 77
Common Symbology Properties 78
Codabar Symbology Properties 79
Code39 Symbology Properties 80
Code 128 Symbology Properties 81
UPC-EAN General Symbology Properties 82
Symbology Names 83
Appendix 3 – Symbology LabelTypes 85
Appendix 4 - Constants 87
MessageBox Constants 87
PlaySound Constants 87
File Attribute Constants 88
Registry Constants 88
Browser Error Constants 89
Glossary 91
Index 92

Preface

All of us at Naurtech Corporation constantly strive to deliver the highest quality products and services to our customers. We are always looking for ways to improve our solutions. If you have comments or suggestions, please direct these to:

Naurtech Corporation
e-mail: support@naurtech.com
Phone: +1 (425) 837.0800

Assumptions

This manual assumes you have a working knowledge of:

- Microsoft Windows user interface metaphor and terminology.
- Stylus based touch screen navigation terminology.
- Basic programming and scripting concepts.
- Dynamic HTML, the browser DOM, and JavaScript.
- Basic operations and requirements of the host applications you want to access with the Naurtech smart client.

Conventions used in this Manual

This manual uses the following typographical conventions:

- All user actions and interactions with the application are in bold, as in **`[Session][Configure]`**

- Any precautionary notes or tips are presented as follows

  Tip: Text associated with a specific tip

- ☞ represents new version specific information
- All text associated with samples is presented as follows.

```
/*alert*/
OS.Alert("Script done.");
```

Additional Documentation

Naurtech Scripting is an integral feature of Naurtech terminal emulation Smart Clients. Please refer to the User's Manual for detailed installation and configuration information. The User's Manual may be downloaded from the "Support" section of our website.

Online Knowledgebase

Although we continually strive to keep this manual up to date, you may find our online support knowledgebase useful for the latest issues, troubleshooting tips and bug fixes. You can access the support knowledgebase from our website at:

www.naurtech.com → Support → Knowledgebase

1.0 Introduction

The Naurtech CETerm Smart Clients provide a robust and flexible environment for Terminal Emulation and Web based applications on a mobile device. Our Clients are available for most Windows CE platforms including CE .NET 4.2, Windows CE 5.0, Windows Mobile 2003, and Windows Mobile 5.

Device tailored versions of our Clients are available for most industrial terminals. These versions integrate the peripherals on each device, such as the barcode scanner, magnetic stripe reader, RFID reader and Bluetooth printer. Naurtech Scripting features provide additional control of these peripherals and simplify tasks such as data collection, validation, and automation.

All Naurtech Clients include one or more Terminal Emulations (TE) and a Web Browser for a natural migration path from legacy text based TE applications to newer Web based applications. We will refer to the clients collectively as CETerm, although the scripting features apply fully to the single emulation products CE3270, CE5250, and CEVT220.

Scripting features can help the transition to web applications and add capabilities to older TE applications. Newer web based applications can be presented in a familiar single-purpose (locked down) configuration which uses keys, the touch screen, or both for user interactions. Please see our “Web Browser Programming Guide” for detailed information on using the Web Browser features.

The Naurtech Scripting features automate and extend our Smart Clients. We use the industry standard JavaScript language with Microsoft JScript additions. JavaScript is the language underlying the most capable and complex functionality available in web applications today. This new class of web applications is sometimes referred to as “Web 2.0” using Asynchronous Javascript and XML (AJAX). CETerm brings this mature and rich language to the TE user to provide more productive TE applications. Scripting can also interact with web browser sessions to enrich and extend existing web applications on the mobile device.

Scripts can be as simple as editing barcode data before sending to a host or as complex as parsing an external XML document, applying an XSLT transformation and returning the result to the host through the TE session. CETerm Automation Objects are provided to give scripts access to the state of CETerm, the TE session, and access to Windows CE operating system functions such as reading a file.

This guide is intended to describe the steps for writing and running scripts and the features provided through the CETerm Automation Objects. Please consult the standard references for details on JavaScript (or JScript) syntax and XML. You may also need to consult standard references for HTML syntax, the browser

Document Object Model (DOM), and other aspects of Dynamic HTML if you are scripting web browser features. Please refer to the Naurtech User's Manual for details on basic usage and configuration of the Naurtech clients.

We hope that our Scripting features will enrich and extend the capabilities of your TE and browser applications. Explore a little deeper and we think you will be amazed at the possibilities for building powerful business applications.

1.1 FEATURE HIGHLIGHTS

Following are some of the special features in Naurtech Scripting

- **JavaScript.** Naurtech uses the industry standard JavaScript scripting language. This powerful language is familiar to programmers and non-programmers world-wide as the core of rich web applications. With JScript, the Microsoft version of JavaScript, additional features are available such as the ability to use ActiveX objects in scripts.

- **On-device Script Editing.** Scripts are saved within the CETerm configuration and can be edited and tested right on the mobile device. Scripts can be imported and exported via text files on the device as well as loaded dynamically from files.

- **Cross Session Scripting.** All Naurtech clients allow up to 4 simultaneous sessions. Scripts can access and control any or all sessions. For example, you could extract text from one TE session and insert it into a different TE session or into a Web application.

- **Automation Objects.** CETerm Automation Objects are available to access and control the state of CETerm, the state of a TE or web browser session, the mobile device, and the Windows CE Operating System. Together these objects provide a rich set of features to simplify routine steps or build complex applications. For example, you can use an automation object to examine the current screen contents to trigger special actions.

- **Enriched Web Browser Applications.** Naurtech Scripting can interact with a web browser session to enrich existing web applications that were not written for a mobile device. For example, key bindings can be added to activate items in the page and scanned barcode or RFID data can be directed to input elements.

- **Workflow Automation.** Scripts can be used to automate routine tasks. The task may be a simple login process or a complex set of steps in your host application.

- **Event Activated Scripts.** There are several events within CETerm that will run associated scripts. For example, when a barcode is read, the script "OnBarcodeRead" will execute and will allow arbitrary processing of the barcode data before it is submitted to the TE or web browser session.

- **Key, Button, and Menu Activated Scripts.** Like most other CETerm actions, scripts can also be tied to any key combination, a toolbar button, or a context menu.

- **Timer Activated Scripts.** Scripts can be scheduled to run at a future time or run periodically.

- **Host Activated Scripts.** Host applications can also invoke scripts using special commands within the TE data stream.

2.0 Getting Started

This section describes some common ways that scripting features can be used within CETerm. Here we describe the JavaScript engine in CETerm and show how to load and edit a script. We also show sample scripts which (1) handle scanner input, (2) auto-login a terminal emulation session, and (3) provide user-specified “hot-spots” on the screen. Only small code “snippets” are shown. For complete details see the later sections of this manual.

2.1 JAVASCRIPT ENGINE

The CETerm JavaScript engine is a full JavaScript environment running in CETerm that provides all the power and familiarity of JavaScript for automating and extending your data collection process. Strictly speaking, CETerm contains the Microsoft JScript engine, which has additional capabilities, but we will refer to it as JavaScript.

The CETerm JavaScript engine is separate from the JavaScript engines which are available in web browser sessions, but the two engines can communicate, exchange data and send commands. Unlike the web browser engine, the CETerm engine runs independently of any TE or browser session and can interact with all sessions. This persistence allows the CETerm engine to maintain state throughout a data collection process.

The CETerm script engine runs as part of the CETerm user interface and when processing a script, the device keys and screen may be unresponsive. Think of the script engine as a virtual user which can examine the screen and send input. There are several techniques to write asynchronous scripts and to show feedback to the user and get user input while a script is running.

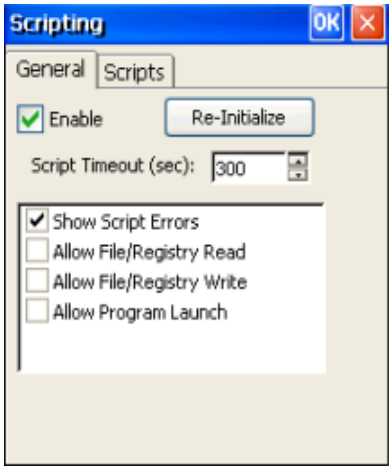
2.2 Enabling Scripting and Editing Scripts

Scripting is disabled by default. To enable scripting, open the configuration dialog

`[Session]->[Configure]->[Options]->[Configure Scripting]`

General Settings

On the **General** tab, check the **Enable** box and check **Show Script Errors**. You may also want to enable file and registry access permission or program launching if you need these features. The **Re-Initialize** button on this tab can be used if you have made changes to the permissions or your scripts and you wish to load the changes. The re-initialization does not take place until the dialog is closed.

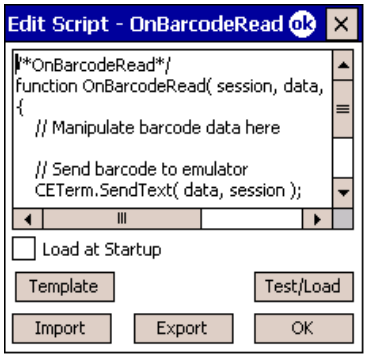


The **Script Timeout** variable limits the duration of script execution. This limit is useful when developing new scripts and as a safeguard against a script with an "infinite loop". A value of 0 will disable the timeout. During execution, a script can modify the timeout value and reset the timer to allow additional execution time.

Editing Scripts

Scripts are edited on the **Scripts** tab. There are 64 script slots. The size of the script in each slot is limited to about 260,000 characters (about one-half megabyte under Windows CE). Scripts can also be loaded dynamically from files. A script slot will usually contain function definitions, which will be loaded into the engine, or executable statements such as function calls which may be bound to a key, toolbar, or menu.

After selecting a script slot and tapping the **Edit** button, an Edit Script dialog will appear. The edit dialog allows **Import** and **Export** of scripts. For initial script development it may be easier to edit on your desktop PC, copy the script to the device, and **Import** the script. Smaller changes are easily made on the device.



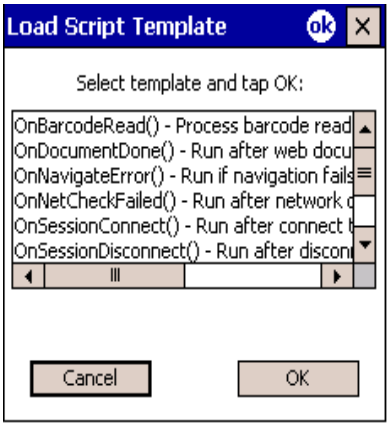
The checkbox **Load at Startup** should be checked for all scripts that contain function definitions that you want to have available in the script engine. The checkbox should **not** be checked for slots that contain scripts that are bound to keys or other activations. **Load at Startup** should be checked for all event handler definitions. All scripts with **Load at Startup** will be loaded into the script engine when it starts with CETerm startup, or when **Re-Initialize** has been pressed on the **General** tab.

After importing or editing a script, you may want to tap the **Test/Load** button. If the script engine was previously enabled, the script will be executed. If the current script is a function definition, it will be checked for correct syntax and will be made available to the script engine. If the current script contains executable

statements or is a function call, it will simulate activating the script. In general, you do not want to use **Test/Load** for executable statements.

Remember to tap **Test/Load** or **Re-Initialize** (with **Load at Startup** checked) after making changes to a script, if you want those changes loaded into the script engine. Also, **Test/Load** will not work if you have just checked **Enable** but not yet accepted the configuration changes.

The **Template** button displays a list of script templates which correspond to the scripting event handlers. Select a template and tap **OK** to have it replace the current contents of the script being edited. The template scripts show some of the ways to use CETerm Automation Objects.



2.3 CETERM AUTOMATION OBJECTS

The CETerm Automation Objects provide access to the running CETerm application, session screens, the Windows CE operating system, and other features. For example the command

```
CETerm.PostIDA( "IDA_SESSION_S1", 0 );
```

within a script would switch CETerm to Session 1 if another session was currently active. Automation Objects can give access to the browser Document Object Model (DOM) of connected web sessions and the text on terminal emulation sessions. The IDA action codes are described briefly in the following section.

The CETerm Automation Objects are similar to ActiveX controls that are used in web pages, but they do not require any special creation operations prior to use. In fact, the same CETerm Automation Objects are accessible from both the CETerm JavaScript engine and the web browser JavaScript engines.

2.4 IDA ACTION CODES

An IDA Action Code is a special value that is used to invoke a device action, program action, or emulator action within the Naurtech Smart Client. IDA Action Codes can invoke special keys under terminal emulation, sound a tone, connect a session, or show the SIP. There are many IDA codes and these are documented in Appendix 1 of this manual. Almost any action which can be invoked by a KeyBar or assigned to a hardware key can be invoked by an IDA code. IDA codes can be submitted to CETerm in several different ways, under both scripting and the web browser.

2.5 ONBARCODEREAD SCRIPT EVENT

CETerm generates several script events during operation. If there is a corresponding event handler defined within the CETerm script engine, then that handler will be invoked. The "OnBarcodeRead" event is a good example. The OnBarcodeRead event handler can intercept and pre-process barcode scan data using the full power of JavaScript before sending the data on to the TE or browser session.

The OnBarcodeRead handler could do something simple, such as pre-pending zero digits for short barcodes, or something complex such as splitting an Automotive Industry Action Group (AIAG) B-10 barcode and putting different parts into different fields on an IBM 5250 emulation screen.

Here is the OnBarcodeRead template that can be loaded in the script edit dialog

```
/* OnBarcodeRead */
function OnBarcodeRead( session, data, source, type, date, time )
{
    // Manipulate barcode data here

    // Send barcode to emulator
    CETerm.SendText( data, session );

    // Return 0 to handle barcode normally
    // Return 1 if handled data here
    return 1;
}
```

This handler simply passes the barcode data on to the current session using the "SendText" method. The return value of 1 tells CETerm not to pass on the barcode data with the usual wedge technique.

The following OnBarcodeRead handler will prefix 3 zeros to any 8 digit barcode and pass other barcodes unchanged

```
/* OnBarcodeRead */
function OnBarcodeRead( session, data, source, type, date, time )
{
    // Prefix zeros to short barcodes
    if (data.length == 8)
    {
        data = "000" + data;
    }
    // Send barcode to emulator
    CETerm.SendText( data, session );

    // Return 0 to handle barcode normally
    // Return 1 if handled data here
    return 1;
}
```

If the OnBarcodeRead handler is defined, it will override any "ScannerNavigate" handler defined in a web page META tag. The following OnBarcodeRead handler will pass the scan on to the ScannerNavigate handler for a web browser in session 2

```
/* OnBarcodeRead */
function OnBarcodeRead( session, data, source, type, date, time )
{
    // Don't process for browser session
    if (session == 2)
    {
        // Return 0 to handle barcode with ScannerNavigate
        return 0;
    }
```

```
    // Prefix zeros to short barcodes
    if (data.length == 8)
    {
        data = "000" + data;
    }
    // Send barcode to emulator
    CETerm.SendText( data, session );

    // Return 1 if handled data here
    return 1;
}
```

The following OnBarcodeRead handler will split any barcode containing an ASCII Linefeed (LF = 0x0A) character and terminated with an ASCII ENQ (ENQ = 0x05) into two parts. The first part is put into the current IBM 5250 field and the second part into the next field and then submitted to the IBM host. This technique is used to login a user with a Code39 barcode in full-ASCII mode. All other barcodes are passed on for normal input

```
/* OnBarcodeRead */
function OnBarcodeRead( session, data, source, type, date, time )
{
    // Look for Full-ASCII Code 39 (LF = 0x0A) to mark Field Exit
    var lfIndex = data.indexOf( "\x0A" );

    if (lfIndex >= 0 )
    {
        var passwordStart = lfIndex + 1;

        // Look for Full-ASCII Code 39 (ENQ = 0x05)
        var enqIndex = data.lastIndexOf( "\x05" );

        if (enqIndex >= 0)
        {
            // NOTE: Using substr to extract user
            // Send User
            CETerm.SendText( data.substr( 0, lfIndex ), session );

            // Send field exit to advance cursor
            CETerm.SendIDA( "IDA_FIELD_EXIT", session );

            // Send Password
            // NOTE: Using substring to extract password
            CETerm.SendText( data.substring( passwordStart,
                                             enqIndex ), session );

            // Submit form
            CETerm.SendIDA( "IDA_ENTER", session );
```

```
            // All scan data handled here
            return 1;
        }
    }

    // Handle scan data in normal way
    return 0;
}
```

The `type` argument to OnBarcodeRead contains the labeltype of the barcode. This labeltype is related to the barcode symbology but usually a little more informative. The values are dependent on the hardware manufacturer but for most devices are the same as the Symbol LABELTYPE. The `type` is a small integer value representing a printable ASCII character (See Appendix 3). The `source` argument is the name of the scanner that read the barcode and is typically unused. The `date` and `time` are text strings representing the time of the read.

2.6 AUTOMATED LOGIN

Automating the host login process is a common task to speed workflow. CETerm contains a Macro record and playback that is usually used for this task. One limitation of the Macro feature is that it will only support a single session auto-connecting when CETerm starts. The scripting feature allows much more power and flexibility for automating the login or any complex or repetitive process.

Most auto-login features are based on a “prompt-and-response” mechanism that waits for text from the host (the prompt) and then sends some text (the response). The “expect” script and “ExpectMonitor” class provide the “prompt-and-response” mechanism within CETerm. The response is usually some simple text, but with the ExpectMonitor, it can be a script itself. The ExpectMonitor is also a good example of using script timers to perform long tasks. The full listing of the “expect” script and “ExpectMonitor” can be found in Section 5.1.

When “expect” is used for auto-login, it is activated within the “OnSessionConnect” event handler. Here is a simple example of an OnSessionConnect handler

```
/* OnSessionConnect */
function OnSessionConnect( session )
{
    // Set login information
    var myusername = "joeuser";
```

```
    var mypassword = "secret";

    var waittime = 8000;    // Milliseconds waiting for each text

    // Only login session 1
    if (session == 1)
    {
        // Look for "login" then "password"
        expect( session, waittime, "Login", myusername + "\r",
                                   "Password", mypassword + "\r" );
    }
}
```

The expect arguments are `session` for the session index, `waittime` for the milliseconds waiting for each expected text, followed by pairs of expected text (prompt) and corresponding action(response). If the action is text, it is simply sent to the host when appropriate. There can be any number of (expected text, action) pairs as arguments. The expected text can be plain text or a regular expression.

For a case-insensitive match of “Login”, an appropriate regular expression could be `/login/i` Regular expressions use the slash character as a delimiter rather than double-quote characters. The ‘i' indicates a case-insensitive match.

A more complex action can contain an anonymous function definition such as

```
var beepMe = function (session) {CETerm.SendIDA("IDA_BEEP_LOUD", 0);
                                 CETerm.SendText("me\r", session ); }
```

Combining these changes into the expect call would give

```
expect( session, waittime, /login/i, beepMe,
                           "Password", mypassword + "\r" );
```

You might wonder why the `SendIDA` call in `beepMe` has a session index of 0 whereas `SendText` has the actual session argument. In this case we know that the beep action is not session specific and does not need to be sent to a specific session. In general, it is always OK to specify a session and it will be ignored for actions that do not require a value.

2.7 CUSTOM SCREEN HOT-SPOTS

A “hot-spot” is an area on a terminal emulation screen that is activated by taping with your finger or the stylus. CETerm supports several pre-defined hot-spots for TE sessions. With scripting, it is possible to define custom hot-spot behaviors. Custom hot-spots use the “OnStylusDown” event handler. Browser sessions do not support the OnStylusDown event because equivalent behavior can be implemented in HTML. You may need to disable the pre-defined hot-spots in CETerm because they will be triggered before a custom hot-spot.

The hot-spot action can depend on the screen contents in an area or simply be tied to a screen area. The following OnStylusDown handler can be loaded from the script templates

```
/* OnStylusDown */
function OnStylusDown( session, row, column )
{
    // Look for custom hot-spot
    var screen = CETerm.Session( session ).Screen;
    var text = screen.GetTextLine( row );
    if (text.match( /beep/i ))
    {
        OS.PlaySound( "default.wav", 0 );
    }
}
```

This hot-spot will play a sound if the line touched contains the word “beep”. The following hot-spots will activate VT function keys if the user touches in the specified rows and columns. In this case, the screen can show a box drawn with VT line drawing characters and text inside each box. With such a display, you can effectively create large glove-friendly on-screen buttons in TE.

```
/* OnStylusDown */
function OnStylusDown( session, row, column )
{
    // Buttons are on rows "start" through "end"
    var buttonrowstart = 9;
    var buttonrowend = 13;
    var IDA = "IDA_NONE";

    // Buttons are "buttonwidth" columns wide
    // Leftmost button is #1
    var buttonwidth = 5;
    var button = Math.floor((column + buttonwidth - 1) /
                                buttonwidth);

    if (row >= buttonrowstart && row <= buttonrowend)
    {
        switch (button) {
```

```
            case 1:   IDA = "IDA_VT_PF1"; break;
            case 2:   IDA = "IDA_VT_PF2"; break;
            case 3:   IDA = "IDA_VT_PF3"; break;
            case 4:   IDA = "IDA_VT_PF4"; break;
            }

            // DEBUG: Uncomment next two lines for testing
            //OS.Alert( "row=" + row + " col=" + column +
            //          " button=" + button + " IDA=" + IDA );

            if (!IDA.match("IDA_NONE"))
            {
                // Send command
                CETerm.PostIDA( IDA, session );
            }
        }
    }
```

You may have noticed by now the use of `PostIDA` in some cases and `SendIDA` in other cases. `SendIDA` is a synchronous activation of an action whereas `PostIDA` is an asynchronous or deferred activation. In general it is always better to use `PostIDA` unless you **must** wait for the action to complete before proceeding in the script. The post action is similar to the “PostMessage” function in Windows programming and the send is similar to the “SendMessage” function. See the CETerm automation object for more details.

3.0 CETerm Automation Objects

This section describes the Automation Objects available to the CETerm script engine. These objects provide access to the running CETerm application, TE session screens, the Windows CE operating system, and other features for developing rich applications.

The automation objects are accessed in a hierarchical manner similar to the Document Object Model (DOM) of a webpage. The two top-level objects are `CETerm` and `OS`. The `CETerm` object provides access to application specific features whereas the `OS` object provides access to generic Operating System (OS) features such as files and the device registry.

Automation objects provide some of the same functionality provided by the Window object in the web browser. For example, the familiar Window methods `alert()` and `setTimeout()` are provided by the `OS.Alert()` and `CETerm.SetTimeout()` methods.

3.1 THE CETERM OBJECT

The `CETerm` object gives access to CETerm features, settings, and session state. This section documents the methods and properties of the `CETerm` object.

Methods

The following methods are available

Method	Action
AbortScript	Abort the currently running script
ClearAllTimers	Clear all SetTimeout and SetInterval timers
ClearInterval	Clear a recurring interval timer
ClearTimeout	Clear a one-time timer
GetProperty	Get a property value
PlaySound	Play a tone or wave file on the device (deprecated)
PlayTone	Play a tone on the device (deprecated)
PostIDA	Send a command to a session (asynchronous)
RunScript	Run a script (called from a web browser only)
SendIDA	Send a command to a session (synchronous)
SendText	Send text to a session
Session	Get a session object
SetInterval	Create a recurring interval script execution timer
SetProperty	Set a property value
SetScriptTimeout	Set the current script execution timeout
SetTimeout	Create a one-time script execution timer

AbortScript( )

Stops the currently executing script.

ClearAllTimers( )

Clear all recurring interval timers and one-time timers.

ClearInterval(intervalTimerID)

Clear the specified recurring interval timer.

ClearTimeout(timerID)

Clear the specified one-time timer.

value = GetProperty(propertyName)

Return the named property value. This may be a device property, application property, or session property. See Appendix 2 for a list of available properties. Returns the JavaScript “undefined” value if the requested property cannot be found.

PlaySound(sound) (deprecated)

Play a tone or wave file on the device. This PlaySound is not the same as the Windows PlaySound of the OS object. This method will accept a wave file name but it will also accept a “tone specifier” as a string to support this legacy feature in the Naurtech Web Browser. New application should use the OS.PlaySound or OS.PlayTone commands. Use the complete file path if the wave file is not in the \Windows directory.

If the handheld device contains a programmable tone generator, the sound parameter may also be a string which defines a sequence of tones to play. The syntax is given below:

“vvfffddd” – where
vv – is the volume 01-10
fff – is the frequency in 10’s of MHz, 000-999
ddd – is the duration in 10’s of milliseconds, 000-999

Multiple tone specifications can be concatenated.

PlayTone(volume, frequency, duration) (deprecated)

Play a tone if supported by the handheld hardware. This method is provided for backward compatibility within the web browser. New application should use OS.Playtone() which provides the same functionality.

volume – is the volume 00 -10 (0 is off, 10 is loudest)
frequency – is the frequency in Hz.
duration – is the duration in milliseconds.

PostIDA(IDASymbolicName, session)

PostIDA submits an IDA action command and directs it to the specified session. Valid session values are 1-4. The special session value of 0 will send the command to the current session. Some IDA commands act at a global level and ignore the session variable. See Appendix 1 for IDA Symbolic Names.

The PostIDA command will return before the action executes. In general, the IDA action will not be applied until after the current script execution ends. We recommend using PostIDA rather than SendIDA. There are only rare situations when SendIDA must be used.

status = RunScript(script)

Run the specified script in the CETerm engine. This method must only be used when the CETerm object is referenced from the web browser script engine. In general, it is better to use PostIDA with an IDA_SCRIPT_xx action to run a pre-defined script from the web browser. To execute a script contained in a string from the CETerm engine use the JavaScript “eval()” method.

SendIDA(IDASymbolicName, session)

SendIDA sends an IDA action command and directs it to the specified session. Valid session values are 1-4. The special session value of 0 will send the command to the current session. See Appendix 1 for IDA Symbolic Names.

The SendIDA method will attempt to complete the action before returning. We recommend using PostIDA rather than SendIDA. There are only rare situations when SendIDA must be used. For example, SendIDA will be needed if you need to invoke IBM field actions, such as Field Exit, between sending text to an IBM session with SendText.

SendText(text, session)

SendText sends a text string to the specified session. Valid session values are 1-4. The special session value of 0 will send the command to the current session. This command is synchronous and CETerm will act on each character before this method returns.

The text string may include IDA symbolic names between backslash characters '\'. The IDA codes will be interpolated as the text is sent. For example, "username\\IDA_FIELD_EXIT\\secretpassword". Note that each backslash has a preceding backslash because it has special meaning within a JavaScript string.

Session (index)

Return the corresponding Session object. Valid index values are 1 to the MaxSession property value. The object is returned even if the session is not connected.

intervalTimerID = SetInterval(scriptExpression, delayMillisec)

Set a recurring interval timer to execute the scriptExpression after each delay of delayMillisec. This method returns an ID that should be saved in a global variable for later use with ClearInterval if needed. Other scripts may run while waiting for this timer. The scriptExpression is a string containing the script, but is commonly a function invocation, such as "myTimerFunction(3, 'alert');"

Timers are especially useful with complex or long-running scripts. Interval timers should be used to perform simple update tasks. One-time timers should be used in preference to interval timers. In general, scripts should perform a short action and exit. With a complex script such as a state-machine, the state can be maintained in global variables and the script re-activated periodically to check for state transitions and perform actions. See the "expect" script and "ExpectMonitor" class in Section 5.1 for an example of the use of a timer.

status = SetProperty(propertyName, propertyValue)

SetProperty will assign the given value to the named property. See Appendix 2 for a list of available properties. The returned status is 0 for success, non-zero for failure.

SetScriptTimeout(millisec)

Set the maximum script execution time. This value may be updated during a running script. If updated, the new timeout will apply starting at the time of the change. A value of 0 will disable the timeout.

timeoutTimerID = SetTimeout(scriptExpression, delayMillisec)

Set a one-time timer to execute the scriptExpression after a delay of delayMillisec. This method returns an ID that should be saved in a global variable for later use with ClearTimeout if needed. Other scripts may run while waiting for this timer. The scriptExpression is a string containing the script, but is commonly a function invocation, such as "myTimerFunction(3, 'alert');"

Timers are especially useful with complex or long-running scripts. Timers can also be used to defer an operation which is not possible within an event handler. One-time timers should be used in preference to interval timers. In general, scripts should perform a short action and exit. With a complex script such as a state-machine, the state can be maintained in global variables and the script re-activated periodically to check for state transitions and perform actions. See the "expect" script and "ExpectMonitor" in Section 5.1 for an example of the use of a timer.

Properties

The `CETerm` object has several application level properties.

Property	Description	Values
ActiveSession	Current active session. (read only)	1-4
MaxSession	Maximum session index. (read only)	4
Message	Returns message object. (read only)	object
TextInput	Return text input object. (read only)	object

3.2 THE SESSION OBJECT

The `Session` object gives access to session state. This section documents the methods and properties of the `Session` object.

Methods

The `Session` object has no methods.

Properties

The `Session` object has several read only properties.

Property	Description	Values
Browser	Returns browser object. (read only)	object
IsConnected	Returns true if session is connected. (read only)	true, false
Screen	Returns screen object. (read only)	object

3.3 THE SCREEN OBJECT

The `Screen` object gives access to a session terminal emulation screen. This section documents the methods and properties of the `Screen` object.

Methods

The following methods are available

Method	Action
GetText	Get all text from start location to end location
GetTextLine	Get all text on a line
GetTextRect	Get a rectangle of text

text = GetText(startRow, startColumn, endRow, endColumn)

Return the requested range of text. Each line will be separated by the TextLineSeparator property value. If the session is not connected the JavaScript null value is returned. End coordinates of -1 will use the maximum valid value.

text = GetTextLine(row)

Return the requested row of text. If the session is not connected the JavaScript null value is returned. The row range is from 1 to the maximum row number.

text = GetTextRect(startRow, startColumn, endRow, endColumn)

Return the requested rectangle of text. Each line fragment will be separated by the TextLineSeparator property value. If the session is not connected the JavaScript null value is returned. End coordinates of -1 will use the maximum valid value.

Properties

The `Screen` object has several read only properties.

Property	Description	Values
Rows	Number of rows in screen. (read only)	1-50
Columns	Number of columns in screen. (read only)	1-132
CursorRow	Current row containing cursor. Setting this value will change the cursor location.	1 to Rows
CursorColumn	Current column containing cursor. Setting this value will change the cursor location.	1 to Columns
TextLineSeparator	Text which separates every line in GetText methods.	Default: nothing

3.4 THE BROWSER OBJECT

The `Browser` object gives access to a web browser session. This section documents the methods and properties of the `Browser` object.

Methods

The following methods are available

Method	Action
AddMetaItem	Add a CETerm <META> element to the current web page.
Navigate	Navigate to specified URL.
RunScript	Run a script in the web browser JavaScript engine.

result = AddMetaItem(target, content)

Add a CETerm <META> tag element to the current web page. This is typically used to add custom <META> elements which define key mappings or other custom behaviors. See the Naurtech Web Browser Programming Guide for documentation on custom <META> tags. Returns 0 for success, non-zero for failure. After adding META elements that change the values of information icons you may need to use CETerm.PostIDA("IDA_INFO_REFRESH", 0) to apply the changes.

result = Navigate(URL)

Navigate the browser session to the specified URL. Returns 0 for success, non-zero for failure.

result = RunScript(script)

Execute the specified script in the browser Javascript engine. Returns 0 for success, non-zero for failure.

Properties

The following read-only properties are available.

Property	Description	Values
Document	Document object of the current web page. The DOM of the page may be examined and altered via this object. WARNING: Use a local variable to hold this reference to minimize memory usage. (read only)	object
DocLoaded	Returns true if document is loaded. (read only)	true, false

3.5 THE MESSAGE OBJECT

The `Message` object provides feedback to the user while a script is running. This object displays a dialog with a text message, an optional progress bar, and an optional script cancellation button. The progress value can be set by the script

as tasks are completed, or it can run at a constant rate to show activity to the user.

Methods

The `Message` object has no methods.

Properties

The `Message` object is controlled through read-write properties. Setting a property will change the message dialog appearance.

Property	Description	Values
AbortButtonVisible	If true, a script abort button is visible. Taping this button will abort the current script execution.	true, false
IsVisible	If true, message dialog is visible.	true, false
Progress	Current progress value in percent.	0 to 100
ProgressRate	Rate of change of progress bar. In units of percent per second.	1 to 100
ProgressRunning	If true, Progress increases at ProgressRate	true, false
ProgressVisible	If true, progress bar is shown	true, false
Text	Text of message.	text
Title	Text in message box title bar.	text
Timeout	Visibility timeout for dialog. After this interval, message dialog is closed. A value of 0 disables this timeout.	0 to 9999

3.6 THE TEXTINPUT OBJECT

The `TextInput` object provides user input in a script. This object displays a dialog with a text message, an input field, a Cancel button and an OK button.

Methods

The following methods are available

Method	Action
GetInput	Get input from the user

result = GetInput( )

GetInput displays the user input dialog. Returns 1 for successful input, 0 if input is canceled by the user, and -1 if there was an error. A default response may be set in the Input property prior to calling GetInput. If no default is desired, be sure to clear Input prior to calling GetInput.

Properties

The `TextInput` has the following properties.

Property	Description	Values
Input	Can be pre-set with default response before calling GetInput. If GetInput returns 1, contains the user input.	text
PasswordMode	If true, input is shown as * characters.	true, false
Prompt	Text prompt message for user.	text
Title	Text in message box title bar.	text

3.7 THE OS OBJECT

The `OS` object provides access to operating system resources such as files and the registry.

Methods

The following methods are available

Method	Action
Alert	Show the user a text message. (synchronous)
Beep	Play a default beep tone.
Exec	Run a separate program.
GetErrorMessage	Get a text error message for a Windows CE error value.
KillProcess	Stop a running process started with Exec.

MessageBox	Display a standard Windows MessageBox.
PlaySound	Play a wave file on the device.
PlayTone	Play a tone on the device.
Sleep	Pause the script execution.
StopSound	Stop an asynchronous playing PlaySound sound.
WaitForProcess	Wait for the specified process to end.

Alert(message)

Show the user a simple text message and wait for them to press OK.

Beep( )

Sound the default Windows beep tone.

status = Exec(programFile, commandLine)

Start the specified program. Returns 0 for success, non-zero for failure. Use GetErrorMessage() to convert a non-zero status to a text message. You should immediately save the property LastExecProcess after a successful Exec call to obtain the process ID for later use in WaitForProcess or KillProcess. The programFile should be a fully qualified filename.

text = GetErrorMessage(error)

Returns a descriptive text message for the specified Windows error.

status = KillProcess(processID)

Attempts to stop the specified process. Returns 0 for success, non-zero for failure. You must obtain the processID from the property LastExecProcess immediately after a successful Exec call.

result = MessageBox(message, title, flags)

Display a standard Windows message box. The title is displayed in the message box title bar. The flags are used to specify the icon and buttons that are visible. Returns a value corresponding to the button pushed to close the dialog. See Appendix 4 for flag definitions.

PlaySound(sound, flags)

Play a wave file on the device. This PlaySound is not the same as the CETerm.PlaySound(). Use the complete file path if the wave file is not in the \Windows directory. The flags control the way the sound is played. See Appendix 4 for flag definitions. Returns true on success, false otherwise.

PlayTone(volume, frequency, duration)

Play a tone if supported by the handheld hardware. New applications should use this method and avoid CETerm.Playtone().

volume – is the volume 00 -10 (0 is off, 10 is loudest)
frequency – is the frequency in Hz.
duration – is the duration in milliseconds.

Sleep(delay)

Delay script execution for specified milliseconds.

result = StopSound( )

Stop any currently playing sound. Returns 0 on success.

status = WaitForProcess(processID, timeout)

Wait for the specified process to exit. Return after timeout milliseconds even if process is still running. Return 0 if process has exited, non-zero for timeout or failure. You must obtain the processID from the property LastExecProcess immediately after a successful Exec call.

Properties

The `os` object has the following properties.

Property	Description	Values
File	Returns the File object. This object provides access to the Windows file systems. (read only)	object

Property	Description	Values
LastError	Returns the last Windows error related to the OS object.	integer
LastExecProcess	Returns the process ID of the last program started via Exec. (read only)	unsigned integer
Registry	Returns the Registry object. This object provides access to the Windows registry. (read only)	object
TickCount	Returns the current tick count from Windows. This provides a millisecond resolution time source. (read only)	unsigned integer

3.8 THE FILE OBJECT

The `File` object provides access to the Windows file system.

Methods

The following methods are available

Method	Action
Copy	Create a copy of an existing file.
CreateDirectory	Create a new directory.
Delete	Delete an existing file.
GetAttributes	Get the attributes of an existing file.
GetOpenFileName	Select a filename with a file Open dialog.
GetSaveFileName	Select a filename with a file Save dialog
Move	Move or rename a file.
Read	Read file contents.
RemoveDirectory	Remove (delete) an existing directory.
SetAttributes	Set the attributes of an existing file.
Write	Write contents to a new or existing file.

status = Copy(existingFile, newFile, overWrite)

Copy an existing file to a new file. If a file already exists with the new file name, copy will fail unless overWrite is true. Returns true for success, false for failure.

Use the `File` properties LastError or LastErrorMessge to get additional error information.

status = CreateDirectory(newDirectory)

Create a new directory. Returns true for success, false for failure.
Use the `File` properties LastError or LastErrorMessge to get additional error information.

status = Delete(filename)

Delete an existing file. Returns true for success, false for failure.
Use the `File` properties LastError or LastErrorMessge to get additional error information.

attributes = GetAttributes(filename)

Return the attributes of the file. Use the `File` properties LastError or LastErrorMessge to get additional error information. See Appendix 4 for attribute definitions.

filename = GetOpenFileName(title, filter)

Return the name of a file specified by the user in an Open file dialog. The title of the dialog should contain descriptive information for the user. For example, “Please select a datafile.” The filter is list of filter pairs. Each pair represents the description of a filter and the file selector wildcards. For a JavaScript file it may look like this: `"JScript File (*.js)\x00*.js\x00\x00"`. Each element of the pair is followed by `"\x00"` as a separator character. The last pair has an additional trailing `"\x00"`. Here is a multiple filter example:

```
"JScript File (*.js)\x00*.js\x00All Files(*.*)\x00*.*\x00\x00"
```

filename = GetSaveFileName(title, filter)

Returns the name of a file specified by the user in a Save file dialog. The title of the dialog should contain descriptive information for the user. For example, “Save file as:”. The filter is list of filter pairs. Each pair represents the description of a filter and the file selector wildcards. For a JavaScript file it may look like this: `"JScript File (*.js)\x00*.js\x00\x00"`. Each element of the pair is followed by `"\x00"` as a separator character. The last pair has an additional trailing `"\x00"`. Here is a multiple filter example:

```
"JScript File (*.js)\x00*.js\x00All Files(*.*)\x00*.*\x00\x00"
```

status = Move(existingFilename, newFileName)

Move or rename an existing file. Returns true for success, false for failure. Use the `File` properties LastError or LastErrorMessge to get additional error information.

contents = Read(filename)

Read entire file and return as contents. The read is an atomic operation which opens the file, reads all contents and closes the file. The File object does not support the concept of an "open" file or reading parts of a file. There must be sufficient memory to hold the entire file contents. There is no error information returned. Use GetAttributes to validate a filename and ensure read access.

status = RemoveDirectory(directoryname)

Delete an existing directory. Returns true for success, false for failure. Use the `File` properties LastError or LastErrorMessge to get additional error information.

status = SetAttributes(filename, attributes)

Sets the attributes of the file. Returns true for success, false for failure. Use the `File` properties LastError or LastErrorMessge to get additional error information. See Appendix 4 for attribute definitions.

status = Write(filename, contents)

Writes contents to the file. Returns true for success, false for failure. Any current contents are first deleted. The write is an atomic operation which opens the file, writes all contents and closes the file. The `File` object does not support the concept of an "open" file or writing parts of a file. To append to a file, use Read to get the current contents and Write the combined current and additional contents. Use the `File` properties LastError or LastErrorMessge to get additional error information.

Properties

The `File` object has the following properties.

Property	Description	Values
LastError	Returns the last error value associated with the File object.	unsigned integer
LastErrorMessage	Returns a text message of the last error associated with the File object. (read only)	text

3.9 THE REGISTRY OBJECT

The `Registry` object provides access to the Windows registry. The registry is a form of database on Windows devices which holds the device configuration. The registry has a hierarchical structure. The "keys" are similar to file folders and the "values" inside a key are similar to files in a folder. Several methods require a "fully qualified" value name which contains the full key hierarchy, begins with a "root" key, and ends with the value name. This fully qualified value name is similar to a file name with the full path.

WARNING: Altering the registry can make your device unusable. Be sure you know the effect of changing values.

Methods

The following methods are available

Method	Action
DeleteKey	Delete an existing key.
DeleteValue	Delete an existing value.
EnumerateKeys	Get all sub-key names of a specified key.
EnumerateValues	Get all value names of a specified key.
FlushKey	Issue the RegFlushKey command.
GetValueType	Get the data type of a value.
ReadValue	Read a value from a key.
ReadValueVBArray	Read a value from a key and return as a Visual Basic array.
WriteValue	Write a value to a key.

status = DeleteKey(keyname)

Deletes an existing key and all values. Returns 0 for success or non-zero for an error. Delete will fail if a key has sub-keys.

status = DeleteValue(keyname, valuename)

Deletes the specified value in an existing key. Returns 0 for success or non-zero for an error.

keylist = EnumerateKeys(keyname)

Return a list of sub-keys of the specified key. See Appendix 4 for key names and definitions. Each sub-key in the list is separated by the current StringSeparator property value.

keylist = EnumerateValues(keyname)

Return a list of values of the specified key. See Appendix 4 for key names and definitions. Each value name in the list is separated by the current StringSeparator property value.

status = FlushKey(keyname)

Performs a Windows CE “RegFlushKey” on the specified key. Some older devices use this to trigger a save of the current registry to persistent memory. Do not use FlushKey unless directed by the device manufacturer. Returns 0 for success, non-zero for error.

type = GetValueType(valuename)

Gets the data type for the specified value. Use a fully qualified value name that starts with a root key. Returns 0 for success or non-zero for an error.

data = ReadValue(valuename)

Read the data from the specified value. Use a fully qualified value name that starts with a root key. Binary values are returned as a list of comma separated

hexadecimal digits. MULTI_SZ strings are separated with the current StringSeparator property value.

data = ReadValueVBArray(valuename)

Read the data from the specified value. Return the data as a Visual Basic array. Use a fully qualified value name that starts with a root key. It is usually best to use the ReadValue method and split the values using JavaScript. In rare circumstances a true array may be needed. It is not possible to return a JavaScript array, but it is easy to convert a Visual Basic array into a JavaScript array using the VBArray object. For example:

```
var valuename = "HKLM\\Comm\\PY21BG1\\Parms\\TcpIp\\DhcpDNS";
var vbarray = new VBArray( OS.Registry.ReadValueVBArray( valuename );
var jsarray = vbarray.toarray();
```

status = WriteValue(valuename, valuedata, datatype)

Write the specified value. Use a fully qualified value name that starts with a root key. WriteValue will create the containing key if it does not exist. Binary values are submitted as a list of comma separated hexadecimal digits. MULTI_SZ strings are separated with the current StringSeparator property value. Returns 0 for success, non-zero for error. See Appendix 4 for root key names and datatype definitions. Common datatypes are “REG_SZ” for a string and “REG_DWORD” for a DWORD value.

Properties

The `Registry` object has the following property.

Property	Description	Values
StringSeparator	Text string that separates MULTI_SZ values. Default: "<|>"	text

4.0 CETerm Script Events

This section describes the script events within the CETerm script engine. These events provide ways to trigger event handlers when various conditions occur in CETerm. The event handlers are arbitrary scripts.

The event model in CETerm uses specific event handler names to bind events to handlers. If the event handler function (e.g., "OnBarcodeRead") is defined in the script engine, it will be executed when the event occurs. There is no special command required to register or bind the function to the event. Event handlers can be re-defined at any time. If the handler is no longer needed, the function can be re-defined as empty.

Event	Fired when…
OnBarcodeRead	Barcode read.
OnDocumentDone	New web page loads.
OnNavigateError	Web navigation fails.
OnNetCheckFailed	Fails to complete network check to host.
OnSessionConnect	Session connects to host.
OnSessionDisconnect	Session disconnects from host.
OnSessionDisconnected	Session disconnected by host.
OnSessionReceive	TE session receives data from host.
OnSessionSwitch	Active session changes.
OnStylusDown	Stylus tap on screen.
OnWakeup	Device resumes after a suspend.

4.1 THE ONBARCODEREAD EVENT

The `OnBarcodeRead` event is fired when a barcode is successfully read. The handler can pre-process the data or check other conditions prior to passing it on to a TE or browser session.

Syntax

```
function OnBarcodeRead( session, data, source, type, date, time )
```

session – index of currently active session
data – barcode data
source – source of barcode. Typically a constant scanner name.
type – labeltype of barcode. See Appendix 3 for values.
date – date of barcode read.
time – time of barcode read.

Example

Several samples for OnBarcodeRead were given in Section 2.5. Following is an example that checks the RF connection before submitting the data to the host. This notifies the user that the barcode was not received by the host and instructs the user to return to RF coverage.

```
/* OnBarcodeRead */
function OnBarcodeRead( session, data, source, type, date, time )
{
    // Check RF status
    var status = CETerm.GetProperty( "device.rf.status" );
    if (status <= 0)
    {
        OS.Alert( "No RF signal detected.\n" +
                  "Barcode discarded.\n" +
                  "Return to RF coverage." );
        // Discard barcode
        return 1;
    }

    // Send barcode to emulator
    CETerm.SendText( data, session );

    // Return 1 if handled data here
    return 1;
}
```

4.2 THE ONDOCUMENTDONE EVENT

The `OnDocumentDone` event is fired when a new webpage has completed loading into a web browser session. The handler can add META tag definitions, examine or alter the Document Object Model (DOM), or add JavaScript methods to the page. This event allows CETerm to enhance a web page for mobile data collection that was not originally designed for such.

Syntax

```
function OnDocumentDone( session )
```

session – index of browser session which completed page load.

Example

This example shows how several META tags can be added to a web page. We will add a "PowerOn" handler, a key remapping, and information item tags to position the RF indicator at a special location. The "PowerOn" handler is often used to navigate to a specific page, such as a login page, when the device resumes. The RF indicator tags will restore a specific location, but could be used to alter the RF indicator location depending on the current page.

```
/* OnDocumentDone */
function OnDocumentDone( session )
{
  var b = CETerm.Session( session ).Browser;

  // Do not process the initial about:blank page
  if (!b.Document.URL.match("about:blank"))
  {
    // Add PowerOn META handler
    b.AddMetaItem( "PowerOn",
                   "Javascript:alert(\"My PowerOn\");" );

    // Insert new JavaScript function
    b.RunScript( "function myf1() {alert(\"F1 Function\");}" );

    // Add Key mapping to inserted function
    b.AddMetaItem( "OnKey_F1", "Javascript:myf1();" );

    // Position RF signal indicator
    b.AddMetaItem( "Signal", "x=195" );
    b.AddMetaItem( "Signal", "y=100" );

    // Update information items for location to take effect
    CETerm.PostIDA( "IDA_INFO_REFRESH", session );
  }
}
```

4.3 THE ONNAVIGATEERROR EVENT

The `OnNavigateError` event is fired if the browser fails to complete a web navigation. Typically, the error handler will redirect the web browser to a "file:" URL on the device for error recovery. This event may fire if the device loses RF coverage during a navigation or the web server crashes. It is a good practice to use the CETerm "Check Network Before Send" feature to validate RF coverage prior to submitting the navigation request and use the `OnNavigateError` for additional error handling.

Syntax

```
function OnNavigateError( session, params )
```

session – index of browser session which failed to navigate.
params – navigation error parameters, including the error number and URL.

The params argument is formatted as URL parameters and has the form:
`error=0x800C0005&url=http://192.168.1.20/application.exe?state=3&scan=0`

Everything after `url=` in the params argument is the URL that failed to navigate, along with all the parameters of that URL. The `error` values are standard Microsoft browser status codes and are defined in Appendix 4.

Example For Windows CE 5.0 devices

This example shows how to redirect a web browser to a static URL on the device.

```
/* OnNavigateError */
function OnNavigateError( session, params )
{
    // Save params in text 3x where x is session index
    // This is required by CE 5.0 devices which do not pass
    // parameters to a "file:" URL.
    CETerm.SetProperty( "app.usertext.3" + session, params );

    // Navigate to static error page
    var b = CETerm.Session( session ).Browser;
    b.Navigate( "file:///error.htm" );
}
```

Note the `CETerm.SetProperty()` call. This method saves the params in persistent memory for later use by the "error.htm" web page. The reason to do this is because the parameters are lost by the Windows CE browser when navigating to a "file:" resource. The error web page can retrieve the params using:

```
var property = "app.usertext.3" + external.sessionindex;
var params = external.CETerm.GetProperty( property );
```

Using the URL, the error page can re-attempt the navigation or decide on other error recovery. Please note that the "User Text x" is used for several purposes in CETerm, including key remapping. Be sure that this use does not collide with other uses in your configuration.

Example For Windows Mobile devices

Handheld devices using Windows Mobile can use a different technique to pass on the params URL. For these devices, the parameters of a "file:" URL are available within the browser. The error parameters can simply be passed on to the static page without using a "User Text x" variable.

```
/* OnNavigateError */
function OnNavigateError( session, params )
{
    // Navigate to static error page
    var b = CETerm.Session( session ).Browser;
    b.Navigate( "file:///error.htm?" + params );
}
```

The error page can access the failed URL parameters using normal techniques to re-attempt the navigation or decide on other error recovery.

```
var params = document.location.search;
```

4.4 THE ONNETCHECKFAILED EVENT

The `OnNetCheckFailed` event is fired if a "Network Check on Send" fails to detect the host system and the Network Check Action is "ida://IDA_SCRIPT_ON_NETCHECKFAILED". Other Network Check Actions are possible, including direct naming of an error URL. See the User Manual for more information. Typically, this error handler will redirect the web browser to a "file:" URL on the device for error recovery.

Syntax

```
function OnNetCheckFailed( session, pendingURL )
```

session – index of browser session attempting navigation.
pendingURL – pending URL for navigation.

The pendingURL is the destination that the user requested but which has been deferred because the host was not contacted. The event handler can re-try the navigation.

Example

This example is nearly identical to the `OnNavigateError` handler except that there is no error number in the pendingURL. This handler shows how to redirect a Windows CE web browser to a static URL on the device.

```
/* OnNetCheckFailed */
function OnNetCheckFailed ( session, pendingURL )
{
    // Save pendingURL in text 3x where x is session index
    // This is required by CE 5.0 devices which do not pass
    // parameters to a "file:" URL.
    CETerm.SetProperty( "app.usertext.3" + session, pendingURL );

    // Navigate to static error page
    var b = CETerm.Session( session ).Browser;
    b.Navigate( "file:///error.htm" );
}
```

See the `OnNavigateError` example above for additional details.

4.5 THE ONSESSIONCONNECT EVENT

The `OnSessionConnect` event is fired when a session initially connects to the specified host. The handler can be used to initiate an automated login using the "expect" script and "ExpectMonitor" class.

Syntax

```
function OnSessionConnect( session )
```

session – index of session which connected.

Example

An example using OnSessionConnect to start the automated login was shown above in Section 2.6 and is repeated below. Please refer to Section 2.6 for details. The "expect" script is discussed in Section 5.1.

```
/* OnSessionConnect */
function OnSessionConnect( session )
{
    // Set login information
    var myusername = "joeuser";
    var mypassword = "secret";

    var waittime = 8000;    // Milliseconds waiting for each text

    // Only login session 1
    if (session == 1)
```

```
    {
        // Look for "login" then "password"
        expect( session, waittime, "Login", myusername + "\r",
                                   "Password", mypassword + "\r" );
    }
}
```

4.6 THE ONSESSIONDISCONNECT EVENT

The `OnSessionDisconnect` event is fired when a session is disconnected by a user action. The handler can be used to switch to another session, exit, or perform other cleanup tasks.

Syntax

```
function OnSessionDisconnect( session )
```

session – index of session which was disconnected by user.

Example

This example will switch to the next connected session when the current session is disconnected. If there are no other connected sessions, then CETerm will exit.

```
/* OnSessionDisconnect */
function OnSessionDisconnect( session )
{
    // Switch to next connected session
    CETerm.SendIDA( "IDA_SESSION_NEXTLIVE", 0 );

    if (CETerm.ActiveSession == session)
    {
        // Still on current session, no others connected.
        CETerm.PostIDA( "IDA_PROGRAM_EXIT", 0 );
    }
}
```

4.7 THE ONSESSIONDISCONNECTED EVENT

The `OnSessionDisconnected` event is fired when a terminal emulation (TE) session is disconnected by the remote host. The handler can be used to attempt to reconnect to the host or perform other cleanup tasks.

Syntax

```
function OnSessionDisconnected( session )
```

session – index of session which was disconnected by remote host.

Example

This example will check for RF coverage and attempt to reconnect if RF is detected.

```
/* OnSessionDisconnected */
function OnSessionDisconnected( session )
{
    // Check RF status
    var status = CETerm.GetProperty( "device.rf.status" );
    if (status <= 0)
    {
        OS.Alert( "No RF signal detected.\n" +
                  "Return to RF coverage and reconnect." );
        return;
    }

    // Attempt to reconnect to host
    CETerm.PostIDA( "IDA_SESSION_CONNECT", session );
}
```

4.8 THE ONSESSIONRECEIVE EVENT

The `OnSessionReceive` event is fired when a terminal emulation session receives data from the connected host. The handler can be used to detect screen content such as an error message and perform a desired action.

Syntax

```
function OnSessionReceive( session, count )
```

session – index of session which received data.
count – count of bytes received.

Example

This example will check the screen content on line 24 looking for an error message. If found, the error is displayed as a popup message.

```
/* OnSessionReceive */
function OnSessionReceive( session, count )
{
    // Get line of text
    var s = CETerm.Session( session ).Screen;
    var line = s.GetTextLine( 24 );

    // Do a regular expression case-insensitive match
    if (line.match( /error/i ))
    {
        OS.Alert( "Error: " + line );
    }
}
```

4.9 THE ONSESSIONSWITCH EVENT

The `OnSessionSwitch` event is fired when the active session changes. The handler can be used to perform a session specific action.

Syntax

```
function OnSessionSwitch( session, previousSession )
```

session – index of session which became active.
previousSession – index of session which was previously active.

Example

This example will reposition the battery information item depending on which browser session is active.

```
/* OnSessionSwitch */
```

```
function OnSessionSwitch( session, previousSession )
{
    var b = CETerm.Session( session ).Browser;

    if (b.DocLoaded)
    {
        var x = (session == 1) ? 195 : 10;
        var y = (session == 1) ? 10 : 100;

        b.AddMetaItem( "Battery", "x=" + x );
        b.AddMetaItem( "Battery", "y=" + y );

        CETerm.PostIDA( "IDA_INFO_REFRESH", 0 );
    }
}
```

4.10 THE ONSTYLUSDOWN EVENT

The `OnStylusDown` event is fired when the user taps a terminal emulation screen with a stylus or finger. This event is only fired if the tap does not activate a standard "touch" feature. All touch features can be disabled in the CETerm configuration. This handler can be used to activate user-defined hot-spots.

Syntax

```
function OnStylusDown( session, row, column )
```

session – index of currently active session
row – row of screen tap
column – column of screen tap.

Example

Several samples for OnStylusDown were given in Section 2.7. Following is an example that starts a barcode scan if the row contains the word "scan". Not all hardware devices support a scan trigger by script. If tapping on an IBM screen, you must tap on an input field in the row, or the focus will not be in an input field when the scan is sent to the session. Of course the "OnBarcodeRead" handler could be used to force the scanned data into a preferred input field.

```
/* OnStylusDown */
function OnStylusDown( session, row, column )
{
    var screen = CETerm.Session( session ).Screen;

   // Get row of text
    var text = screen.GetTextLine( row );
```

```
    // Look for "scan" as case-insensitive match
    if (text.match( /scan/i ))
    {
        CETerm.PostIDA( "IDA_SCAN_TRIGGER", 0 );
    }
}
```

4.11 THE ONWAKEUP EVENT

The `OnWakeup` event is fired when the device resumes after suspending. The handler can be used to perform any action, such as waiting for RF coverage or switching to a specific session.

Syntax

```
function OnWakeup( )
```

Example

This example will wait for RF coverage to resume and sound a tone when it is available. While waiting, a “tic” sound will be made periodically to indicate the check. This sample is more complex than needed, but it illustrates how to use global variables and timers to periodically check state.

```
/* RFSoundOnConnect */

// Global control variables
var RFWakeupSoundTimerID = 0;
var RFWakeupSoundContinue = 0;
var RFWakeupSoundInterval = 200;    // milliseconds
var RFWakeupSoundCountMaximum = 50; // 50*200 = 10 seconds
var RFWakeupSoundCount = 0;


function OnWakeup()
{
  // Start with wakeup event
  RFWakeupSoundStart();
}

// Function to start RF check
function RFWakeupSoundStart()
{
  if (!RFWakeupSoundContinue)
```

```
  {
    RFWakeupSoundContinue = 1;
    if (RFWakeupSoundTimerID != 0)
    {
      // Stop and clear any previous check
      CETerm.ClearTimeout( RFWakeupSoundTimerID );
      RFWakeupSoundTimerID = 0;
    }
    RFWakeupSoundCount = 0;

    // Schedule first RF check
    RFWakeupSoundTimerID = CETerm.SetTimeout(
                             "RFWakeupSoundOnTimer();",
                             RFWakeupSoundInterval );
  }
}

// Function to check RF and notify user
function RFWakeupSoundOnTimer()
{
  RFWakeupSoundTimerID = 0;
  RFWakeupSoundCount++;

  // Get and check info
  var rfStatus = CETerm.GetProperty( "device.rf.status" );

  if (rfStatus > 0)
  {
    // RF detected
    // Delayed playsound, increase delay for WEP if needed
    CETerm.SetTimeout( "RFWakeupSoundPlay();", 100 );
    RFWakeupSoundContinue = 0;
  }
  else if (RFWakeupSoundCount > RFWakeupSoundCountMaximum)
  {
    // Failed to get RF, show failure message.
    OS.Alert( "Failed to detect RF signal.\n" +
              "Return to coverage area." );
    RFWakeupSoundContinue = 0;
  }

  if (RFWakeupSoundContinue)
  {
    if (!(RFWakeupSoundCount % 5))
    {
      // Play "tick" sound while check is running.
      CETerm.PlaySound( "MenuPop" );
    }

    // Schedule next RF check
    RFWakeupSoundTimerID = CETerm.SetTimeout(
                             "RFWakeupSoundOnTimer();",
                             RFWakeupSoundInterval );
```

```
  }
}


function RFWakeupSoundPlay()
{
  // Select any wave file on device for notification.
  CETerm.PlaySound( "infbeg" );
}
```

5.0 Scripting Techniques and Tips

This section describes ways that scripting can extend the capabilities of CETerm. Tips for script development are also presented.

5.1 EXPECT AND EXPECTMONITOR FOR AUTOMATING TASKS

The “expect” script and “ExpectMonitor” class provide a general purpose “prompt-and-response” tool. Using “expect” for automated login was described in Section 2.6. Here we provide the complete listing of the scripts and discuss other options for use.

5.1.1 Expect Script

The “expect” script illustrates a couple of powerful JavaScript constructs. Even though the expect function has 4 defined arguments in the function declaration, it is possible to pass an unlimited number of arguments. All arguments are accessible through the special “arguments” variable. This script also shows the object-oriented aspects of JavaScript by creating a new ExpectMonitor class.

```
/* expect */
//
// This script will "expect" a text prompt on the screen and
// respond with text or action.
//
// Syntax: expect( session, timeout,
//                 expectedText, response
//                 [,expectedText2, response2] )
//
// session is the session index
// timeout is the wait interval for each text in milliseconds
// expectedText can be a string or regular expression
// Response can be a text response or a function

function expect( session, timeout, expectedText, response )
```

```
{
    // Build array from arguments
    // This technique will accumulate any
    // number of expect/response pairs
    var TargetResponseArray = [];
    for (var i=2; i < arguments.length; i++)
    {
        TargetResponseArray.push( arguments[i] );
    }

    // Create an ExpectMonitor class that manages the actions
    var EM = new ExpectMonitor( session, timeout,
                                TargetResponseArray );

    // Set optional ExpectMonitor behaviors
    //EM.silent = true;
    //EM.OnDone = function (success) { OS.Alert( "Done." ) };

    // Start check
    EM.Start();
}
```

5.1.2 ExpectMonitor Class

The “ExpectMonitor” class illustrates the use of a prototype in JavaScript. This class also manages all instances of itself to restrict the number of objects that can be created.

```
/* ExpectMonitor */
//
// ExpectMonitor class
//
// The ExpectMonitor class manages the expect/action
// sequence for a session.
// Only one ExpectMonitor is allowed per session.
//


function ExpectMonitor( session, timeout, targetactions )
{
    // Validate session
    if (session < 1 || session > 4)
    {
        return null;
    }

    this.session = session;
    this.timeout = timeout;
    this.args = targetactions;
```

```
    this.state = 0;
    this.timer = null;
    this.checkCount = 0;
    this.totalCheckCount = 0;
    this.maxCheckCount = this.timeout / this.checkDelta;

    // Abort any existing object
    if (ExpectMonitor.Instances[this.session] != null)
    {
        ExpectMonitor.Instances[this.session].Abort();
    }

    // Record this instance in the global array
    ExpectMonitor.Instances[this.session] = this;
}


function ExpectMonitor_Check()
{
    // Clear timer id
    this.timer = null;

    // If something to check for, check it.
    var target = this.args[this.state];

    if (target != null)
    {
        // Get all screen text
        var screenText =
           CETerm.Session(this.session).Screen.GetText(1,1,-1,-1);

        if (screenText != null && screenText.match( target ))
        {
            // Found match
            var action = this.args[this.state + 1];
            this.checkCount = 0;

            if (action != null)
            {
                // Check action
                if (typeof action == "function")
                {
                    // Run function action
                    // Pass session number as argument
                    action( this.session );
                }
                else if (typeof action == "string")
                {
                    // Send text to session
                    CETerm.SendText( action, this.session );
                }
                else if (!this.silent)
                {
```

```
                    OS.Alert("Unknown action type for expect.");
                }
            }

            // Check if another match expected
            this.state +=2;
            target = this.args[this.state];

            if (target != null)
            {
                // Schedule next check
                this.Schedule();
            }
            else
            {
                // Done with this expect.
                // Run any post-execution actions
                if (typeof this.OnDone == "function")
                {
                    this.OnDone( true );
                }
            }
        }
        else
        {
            // No match, schedule again
            if (this.checkCount++ < this.maxCheckCount)
            {
                this.Schedule();
            }
            else
            {
                if (!this.silent)
                {
                    OS.Alert( "Expect failed to find text \"" +
                              target + "\"" );
                }
                if (typeof this.OnDone == "function")
                {
                    // Done but failed
                    this.OnDone( false );
                }
            }
        }
    }
}


function ExpectMonitor_Schedule()
{
    // Schedule next check
    var script = "ExpectMonitor.Instances[" +
                    this.session + "].Check()";
```

```
    this.timer = CETerm.SetTimeout( script, this.checkDelta );
}


function ExpectMonitor_Start()
{
    // Cleanup first in case restarted
    this.Abort();

    // Initialize state
    this.state = 0;
    this.checkCount = 0;

    this.Check();
}


function ExpectMonitor_Abort()
{
    // Stop any timer
    if (this.timer != null)
    {
        CETerm.ClearTimeout( this.timer );
        this.timer = null;
    }

    // Set state to beyond reasonable range
    this.state = 1000;
}


// Method definitions
ExpectMonitor.prototype.Check = ExpectMonitor_Check;
ExpectMonitor.prototype.Schedule = ExpectMonitor_Schedule;
ExpectMonitor.prototype.Start = ExpectMonitor_Start;
ExpectMonitor.prototype.Abort = ExpectMonitor_Abort;

ExpectMonitor.prototype.OnDone = null;


// Check every 200 milliseconds
ExpectMonitor.prototype.checkDelta = 200;
// About 10 seconds for each text check
ExpectMonitor.prototype.maxCheckCount = 50;
// Allow messages
ExpectMonitor.prototype.silent = false;


// Class statics
ExpectMonitor.Instances = [];
```

5.1.3 Automating Tasks with Expect

Any routine prompt-and-response task can be automated with “expect”. Examples may be navigating through a hierarchy of menus or closing an order for shipping. In any case, you identify text to find on the screen and the user input to take you to the next screen. Here is a simple menu traversal:

```
// Traverse menu
expect( CETerm.ActiveSession, 8000,
        "3. Applications", "3\r",
        "2. Inventory", "2\r",
        "2. Put Back", "2\r" );
```

This script can be entered into any script slot and bound to a key combination for activation. You must also load the “expect” and “ExpectMonitor” in a script slot which is marked “Load at Startup” so that the functions are available for use.

5.2 PRESENTING VISUAL FEEDBACK DURING SCRIPT EXECUTION

The Message object can be displayed during script execution when you want to provide a visual indication of script progress. The Message object is asynchronous and a script can continue running while it is displayed. This is unlike the OS.Alert() message which stops script execution and requires user confirmation. There is only one Message object within CETerm and you can change the Message properties within any script.

WARNING: You must exercise caution when using the Message box to avoid leaving it visible after a script is done. You may want to provide a cleanup script that can be activated by the user to be sure the message is hidden.

Following is an example of using the Message box. This message will display itself for 5 seconds and then disappear.

```
/* Show message for 5 seconds */
var m = CETerm.Message;
m.Text = "Processing data, please wait.";
m.Timeout = 5;
m.AbortButtonVisible = true; // does nothing because script exits
m.Progress = 0;
m.ProgressRunning = true;
m.ProgressVisible = true;
m.ProgressRate = 20;
```

```
m.IsVisible = true;
```

You may want to update the progress bar directly while processing data. Here is an example.

```
/* Update progress and message during processing */
var m = CETerm.Message;
m.Text = "Processing data, please wait.";
m.Timeout = 0;
m.AbortButtonVisible = false;
m.ProgressRunning = false;

// Do some work
m.Progress = 0;
m.IsVisible = true;
OS.Sleep( 2000 );  // Simulate work delay

// Update
m.Progress = 20;
m.Text = "Finding addresses, please wait.";
OS.Sleep( 2000 );  // Simulate work delay

// Update
m.Progress = 50;
m.Text = "Sorting addresses, please wait.";
OS.Sleep( 2000 );  // Simulate work delay

// Update
m.Progress = 90;
m.Text = "Almost done, please wait.";
OS.Sleep( 2000 );  // Simulate work delay

// Done
m.IsVisible = false;
```

5.3 GETTING USER INPUT TO A SCRIPT

The TextInput object can get user input for a script. Here is an example for getting a password.

```
/* Get password from user */
var t = CETerm.TextInput;

t.Title = "Warehouse Management";
t.Prompt = "Please enter your password:";
t.PasswordMode = true;
t.Input = "";  // Clear current password
```

```
var s = t.GetInput();
if (s == 1)
{
   OS.Alert( "Password is " + t.Input );
   t.Input = "";  // Clear password
}
else
{
   OS.Alert( "Failed to get password." );
}
```

5.4 RUNNING AN EXTERNAL PROGRAM

It is possible to start an external program from the CETerm script engine. You can wait for the program to finish or allow it to run independently. Usually you will run a program then return to CETerm when it exits.

Here is an example to start the stylus calibration. Note that the arguments depend on whether your device is Window CE or Windows Mobile.

```
/* Stylus Calibration */

// TODO: Uncomment the line for your device

// For Windows CE 5.0 devices
OS.Exec( "\\Windows\\ctlpnl.exe", "cplmain.cpl,9,1" );

// For Windows Mobile 5 devices
//OS.Exec( "\\Windows\\ctlpnl.exe", "cplmain.cpl,7,0" );
```

5.5 USING TIMERS TO RUN SCRIPTS

Script execution timers are useful for several tasks. They can be used to:

1. Defer an action which is not possible in an event handler.
2. Perform an action periodically.
3. Provide an asynchronous script execution.
4. Split up a long running task.

We have already shown how the timer is used with the ExpectMonitor class and task automation in Section 5.1. Event handlers should be limited to a small amount of processing. If more processing is needed, it is best to schedule that processing with SetTimeout() and allow the event handler to exit.

The following example will save data from memory to a flash file whenever a particular URL is loaded.

```
/* OnDocumentDone */
function OnDocumentDone( session )
{
    var b = CETerm.Session( session ).Browser;

    if (b.Document.URL.match( /InventorySave/ ))
    {
        // Resume online inventory, and save cached
        // data to file in background.
        CETerm.SetTimeout( "BackgroundSave(" + session + ");", 10 );
    }
}

/* BackgroundSave */
function BackgroundSave( session )
{
    var d = new ActiveXObject( "Microsoft.XMLDOM" );
    d.loadXML(
      "<?xml version=\"1.0\"?><Books>" +
      "<Book QTY=\"10\"><Title>Beginning XML</Title></Book>" +
      "<Book QTY=\"2\"><Title>Mastering XML</Title></Book>" +
      "</Books>");

    if (!OS.File.Write( "\\FlashDisk\\inventory.xml", d.xml ))
    {
        OS.Alert( "Failed to save inventory." );
    }
}
```

5.6 ACCESSING A FILE

The File automation object provides basic access to the Windows CE filesystem. It supports whole-file read and write, but does not support the concept of an “open” file with piecewise read or write. You can also create and delete file directories.

This example shows how to append to an existing file by using a combination of read and write.

```
/* AppendToFile */
function AppendToFile( filename, addedContent )
{
    var status = false;
```

```
    var F = OS.File;

    // Check if file exists
    var attributes = F.GetAttributes( filename );
    if (attributes != 0xFFFFFFFF)
    {
        var content = F.Read( filename );
        status = F.Write( filename, content + addedContent );
    }
    else
    {
        status = F.Write( filename, addedContent );
    }

    return status;
}
```

5.7 ACCESSING THE REGISTRY

The registry on a Windows CE device is a form of database which contains most of the device configuration. The `Registry` automation object allows you to read, write and delete registry keys and values.

WARNING: Altering the registry can make your device unusable. Be sure you know the effect of changing values and accept the responsibility.

The registry has a hierarchical structure. The "keys" are similar to file folders and the "values" inside a key are similar to files in a folder. Several `Registry` methods require a "fully qualified" value name which contains the full key hierarchy, begins with a "root" key, and ends with the value name. This fully qualified value name is similar to a file name with the full path.

The following example creates a new key and value and confirms that it can be read.

```
/* NewRegistryDWORD */
function NewRegistryDWORD( keyname, valuename, valuedata )
{
    var status = false;
    var R = OS.Registry;

    // Check if file exists
    var fullyQualifiedKey = "HKEY_LOCAL_MACHINE\\" +
                    keyname + "\\" + valuename;

    if (!R.WriteValue( fullyQualifiedKey, valuedata, "REG_DWORD" ))
    {
```

```
            // Check if can read value
            var readdata = R.ReadValue( fullyQualifiedKey );
            if (readdata == valuedata)
            {
                status = true;
            }
        }

        if (!status)
        {
            OS.Alert( "Failed to confirm write of " +
                      fullyQualifiedKey );
        }

        return status;
    }
```

NewRegistryDWORD may be used as follows.

```
// Write a new value
NewRegistryDWORD( "SOFTWARE\\Naurtech\\Test", "TestDword", 510 );
```

5.8 WRITING EFFICIENT SCRIPTS

Good programming practices should be used when developing scripts for CETerm. In general, it is important to conserve memory, minimize script compilations, and limit execution times. Please refer to a JavaScript programming book for more information.

5.8.1 Use Local Variables

Whenever possible, use local variables within functions and declare them with the var keyword, like this:

```
var status;
var message = "hello";
var i, j, k;
```

If you fail to use the var keyword, then JavaScript automatically creates a global variable with that name if it has not already been declared outside a function.

JavaScript uses “garbage collection” to reclaim memory no longer in use. Memory occupied by global variables may never be reclaimed, whereas local variable memory can be reclaimed after a function call completes. Because the

JavaScript engine in CETerm is not reset frequently like a browser JavaScript engine, it is more likely that poor programming practices could exhaust memory.

5.8.2 Encapsulate Code in Functions

Whenever possible, put multiple script actions within a function. This should minimize compilations and make it easier to use local variables as described above. For example, the following actions could be in a script which is bound to a key-combination:

```
CETerm.SetProperty( "session1.scanner.upca.enabled", true );
CETerm.SetProperty( "session1.scanner.msi.enabled", false );
CETerm.SetProperty( "session1.scanner.pdf417.enabled", false );
CETerm.PlayTone( 8, 2000, 200 );
CETerm.PlayTone( 8, 1500, 200 );
CETerm.PostIDA( "IDA_SCAN_APPLYCONFIG", 0 );
```

Or, the actions could be in a function which is loaded with “Load at Startup”

```
function enableUPCA()
{
    CETerm.SetProperty( "session1.scanner.upca.enabled", true );
    CETerm.SetProperty( "session1.scanner.msi.enabled", false );
    CETerm.SetProperty( "session1.scanner.pdf417.enabled", false );
    CETerm.PlayTone( 8, 2000, 200 );
    CETerm.PlayTone( 8, 1500, 200 );
    CETerm.PostIDA( "IDA_SCAN_APPLYCONFIG", 0 );
{
```

and the function call, in a separate script, could be bound to the key-combination:

```
enableUPCA();
```

Using the later approach, the function is only compiled once, not each time the key is pressed. In general, separating the function definitions from the invocation is a good practice.

5.8.3 Limit Execution Time

Because the script engine acts like a “virtual user”, when a script is executing, CETerm will seem unresponsive. You cannot have a script running continuously. However, using events and timers, you can accomplish most tasks.

Do not disable the “Script Timeout” unless you are sure your script will not enter an infinite loop.

5.9 DEBUGGING SCRIPTS

All but the most trivial script will require some degree of debugging.

5.9.1 Show Script Errors

The first step is to enable “Show Script Errors”. This will enable a popup message for compilation and runtime errors. Compilation errors will usually be seen when new scripts are added or upon script engine startup. It may not be clear which script loaded at startup contains the error. In this case you may need to open the edit dialog for each script and tap the “Test/Load” button to identify the bad script.

The compilation error looks like this:

```
Microsoft JScript compilation error
[Line: 15 Col: 8] Expected ‘)’
        OS.Alert( message );
```

Notice that the line of script presented looks OK. In this case, the missing ‘)’ is on the previous line of script, but the error is detected as the compiler reaches column 8 of this line and encounters the ‘O’. Be sure to look around the indicated location to identify the source of the error.

A runtime error may be seen at startup if a script is performing some initialization, or it may be seen while using CETerm. It can be difficult to identify the source of the error if the script was fired by an event or timer. Most often, a runtime error can be prevented by “defensive coding” where you are sure to check the validity of arguments and object references.

The runtime error looks like this:

```
Microsoft JScript runtime error
[Line: 14 Col: 9] Object doesn’t support this property
or method.
```

Unfortunately, the JScript engine does not return the source code line for a runtime error. You must manually examine your scripts at the specified location for a clue about the problem.

5.9.2 OS.Alert()

Because there is no JScript debugger on the Windows CE device, the tried-and-true debugging tool is “OS.Alert(message)”. Experienced programmers will recognize this as the “write(6,100)”, “printf” or “MessageBox” technique.

The basic idea is to sprinkle “OS.Alert()” calls through your code to track program flow and variable values. It can be tedious, but it’s easy to do and easy to remove the OS.Alert() calls by preceding them with comment characters.

Alternatively, you can define a Debug() method and sprinkle it through your code. This makes it easier to enable or disable debugging.

```
var globalDebugLevel = 0;

function Debug( message )
{
    if (globalDebugLevel > 0)
    {
        OS.Alert( message );
    }
}
```

Appendix 1 - IDA Action Codes

Many IDA codes apply only to a Terminal Emulation session. Some IDA codes can only be used in restricted circumstances, such as IDA_URL.

Symbolic Name	Friendly Name	Description
IDA_BEL	Bell	
IDA_BS	Backspace	
IDA_HT	Horizontal Tab	
IDA_TAB	Tab	
IDA_LF	Linefeed	
IDA_VT	Vertical Tab	
IDA_FF	Form Feed	
IDA_CR	Carriage Return	
Printable ASCII		
IDA_SPACE	<Space>	
IDA_EXCLAMATION_MARK	!	
IDA_DOUBLE_QUOTE	"	
IDA_NUMBER_SIGN	#	
IDA_DOLLAR_SIGN	$	
IDA_PERCENT	%	
IDA_AMPERSAND	&	
IDA_SINGLE_QUOTE	'	
IDA_LEFT_PAREN	(	
IDA_RIGHT_PAREN	)	
IDA_ASTERISK	*	
IDA_PLUS	+	
IDA_COMMA	,	
IDA_HYPHEN	-	
IDA_PERIOD	.	
IDA_SLASH	/	
IDA_0	0	
IDA_1	1	
…	…	
IDA_9	9	
IDA_COLON	:	
IDA_SEMICOLON	;	
IDA_LESS_THAN	<	
IDA_EQUAL	=	
IDA_GREATER_THAN	>	
IDA_QUESTION_MARK	?	

Symbolic Name	Friendly Name	Description
IDA_AT	@	
IDA_A	A	
IDA_B	B	
…	…	
IDA_Z	Z	
IDA_LEFT_BRACKET	[	
IDA_BACKSLASH	\	
IDA_RIGHT_BRACKET	]	
IDA_CARET	^	
IDA_UNDERSCORE	_	
IDA_BACKTICK	`	
IDA_a	a	
IDA_b	b	
…	…	
IDA_z	z	
IDA_LEFT_BRACE	{	
IDA_PIPE	|	
IDA_RIGHT_BRACE	}	
IDA_TILDE	~	
IDA_DEL	DEL	
C1 ASCII Controls		
IDA_IND	Index	
IDA_NEL	Next Line	
IDA_HTS	Horiz Tab Set	
IDA_RI	Reverse Index	
IDA_SS2	Single Shift 2	
IDA_SS3	Single Shift 3	
IDA_DCS	Device Ctrl Str	
IDA_PU1	Private Use One	
IDA_PU2	Private Use Two	
IDA_CSI	Ctrl Seq Intro	
IDA_ST	String Term	
IDA_OSC	OS Command	
IDA_PM	Private Msg	
IDA_APC	App Prog Cmd	
Internal Actions (TE only)		
IDA_UPDATE_CURSOR	Update Cursor	

Symbolic Name	Friendly Name	Description
IDA_INHIBIT_UPDATE	Inhibit Update	Don't update display
IDA_UNINHIBIT_UPDATE	Uninhibit Update	Allow display update
IDA_UPDATE	Update	Force display update
IDA_INHIBIT_SEND	Inhibit Send	VT buffer characters
IDA_UNINHIBIT_SEND	Uninhibit Send	VT stop buffering
IDA_SEND_PENDING	Send Pending Chars	VT send buffered chars
Program Actions		
IDA_PROGRAM_ABOUT	Program About	Display About dialog
IDA_PROGRAM_EXIT	Program Exit	Exit program
IDA_PROGRAM_EXITSILENT	Program Exit Silent	Exit program silently
IDA_PROGRAM_HELP	Program Help	Display Help
IDA_SUSPEND_DEVICE	Suspend Device	Enter suspend state
IDA_BLUETOOTH_DISCOVERY	Bluetooth Discovery	Start discovery
IDA_WARMBOOT	Warm Boot	Warm boot device
IDA_COLDBOOT	Cold Boot	Cold boot device
IDA_MENU_TOPBOTTOM	Menu Top/Bot	Toggle menu location
IDA_MENU_TOGGLEHIDE	Menu Toggle	Toggle menu visibility
IDA_TOOLBAR_TOGGLE	ToolBar Toggle	Toggle toolbar visibility
IDA_START_TOGGLEHIDE	Start Menu Toggle	Toggle Start visibility
IDA_MENUBAR_TOGGLEHIDE	MenuBar Toggle	Toggle menubar visibility
IDA_SESSION_TOGGLECON	Connect/Discon	Toggle session connection
IDA_SESSION_CONFIGURE	Configure	Configure session
IDA_SESSION_CONNECT	Connect	Connect session
IDA_SESSION_DISCONNECT	Disconnect	Disconnect session
IDA_SESSION_NEXT_LIVE	Next Live Session	Switch to next live session
IDA_SESSION_PASSWORD	Password	Session password dialog
IDA_SESSION_PREV	Prev Session	Switch to previous session
IDA_SESSION_NEXT	Next Session	Switch to next session
IDA_SESSION_DISCON_ALL	Disconnect All	Disonnect all sessions
IDA_SESSION_S1	Session 1	Switch to session 1
IDA_SESSION_S2	Session 2	Switch to session 2
IDA_SESSION_S3	Session 3	Switch to session 3

Symbolic Name	Friendly Name	Description
IDA_SESSION_S4	Session 4	Switch to session 4
IDA_TOOLBAND_HIDE	Hide ToolBar	Hide full Toolbar
IDA_TOOLBAND_TOGGLEHIDE	Toggle ToolBar	Toggle Toolbar visibility
IDA_KEYBAR_HIDE	Hide KeyBar	Hide KeyBar
IDA_KEYBAR_TOGGLEHIDE	KeyBar Toggle	Toggle KeyBar visibility
IDA_KEYBAR_LEFT	Prev KeyBar	Switch to previous KeyBar
IDA_KEYBAR_RIGHT	Next KeyBar	Switch to next KeyBar
IDA_KEYBAR_SEPARATOR	--Separator--	Separator for KeyBar
IDA_KEYBAR_NONE	(Empty)	No action placeholder
IDA_HSCROLL_HIDE	HScroll Hide	
IDA_HSCROLL_VISIBLE	HScroll Show	
IDA_HSCROLL_TOGGLEHIDE	HScroll Toggle	
IDA_HSCROLL_PLUSON	HScroll Right One	
IDA_HSCROLL_MINUSONE	HScroll Left One	
IDA_HSCROLL_PLUSHALF	HScroll Right Page	
IDA_HSCROLL_MINUSHALF	HScroll Left Page	
IDA_HSCROLL_PLUSEND	HScroll Right End	
IDA_HSCROLL_MINUSEND	HScroll Left End	
IDA_VSCROLL_HIDE	VScroll Hide	
IDA_VSCROLL_VISIBLE	VScroll Show	
IDA_VSCROLL_TOGGLEHIDE	VScroll Toggle	
IDA_VSCROLL_PLUSONE	VScroll Up One	
IDA_VSCROLL_MINUSONE	VScroll Down One	
IDA_VSCROLL_PLUSHALF	VScroll Up Page	
IDA_VSCROLL_MINUSHALF	VScroll Down Page	
IDA_VSCROLL_PLUSEND	VScroll Up End	
IDA_VSCROLL_MINUSEND	VScroll Down End	
IDA_FONT_PLUS	Font Inc	Increase font size
IDA_FONT_MINUS	Font Dec	Decrease font size
IDA_TOGGLE_FONT_BOLD	Font Bold	
IDA_SMARTPAD_OPEN	SmartPad Show	
IDA_SMARTPAD_CLOSE	SmartPad Hide	
IDA_SMARTPAD_TOGGLEHIDE	SmartPad Toggle	
IDA_SLEEP_10	Sleep 10msec	
IDA_SLEEP_50	Sleep 50msec	

Symbolic Name	Friendly Name	Description
IDA_SLEEP_200	Sleep 200msec	
IDA_SLEEP_1000	Sleep 1sec	
IDA_SLEEP_5000	Sleep 5sec	
IDA_SLEEP_20000	Sleep 20sec	
IDA_SLEEP_100000	Sleep 100sec	
IDA_SCAN_RESUME	Scan Resume	Allow scanning
IDA_SCAN_SUSPEND	Scan Suspend	Suspend scanning
IDA_SCAN_TRIGGER	Scan Trigger	Soft trigger scanner
IDA_MACRO_OPEN	Macro Show	Show Macro Tool
IDA_MACRO_CLOSE	Macro Hide	Hide Macro Tool
IDA_MACRO_TOGGLEHIDE	Macro Toggle	Toggle Macro Tool hiding
IDA_MACRO_RECORD	Macro Record	Start Macro record
IDA_MACRO_STOP	Macro Stop	Stop Macro record
IDA_MACRO_PLAY	Macro Play	Replay Macro
IDA_PRINT_SCREEN	Print Screen	Print current screen
IDA_OIA_HIDE	OIA Hide	Hide IBM OIA bar
IDA_OIA_VISIBLE	OIA Show	Show IBM OIA bar
IDA_OIA_TOGGLEHIDE	OIA Toggle	Toggle OIA bar visibility
General IBM and VT Actions		
IDA_PF1	F1	(Not VT PF1)
IDA_PF2	F2	(Not VT PF2)
IDA_PF3	F3	(Not VT PF3)
IDA_PF4	F4	(Not VT PF4)
…	…	
IDA_PF24	F24	
IDA_HOME	Home	
IDA_DOWN	Down	
IDA_UP	Up	
IDA_LEFT	Left	
IDA_RIGHT	Right	
IDA_ENTER	Enter	
IBM Actions		

Symbolic Name	Friendly Name	Description
IDA_IBM_HOME	IBM Home	
IDA_DELETE	Delete	
IDA_INSERT_ON	Insert On	
IDA_INSERT_OFF	Insert Off	
IDA_INSERT_TOGGLE	Insert Toggle	
IDA_ATTN	Attn	
IDA_CLEAR	Clear	
IDA_CURSOR_SELECT	Cursor Select	
IDA_DUP	DUP	
IDA_ERASE_EOF	Erase EOF	
IDA_ERASE_INPUT	Erase Input	
IDA_FIELD_MARK	Field Mark	
IDA_NEWLINE	Newline	
IDA_PA1	PA1	
IDA_PA2	PA2	
IDA_PA3	PA3	
IDA_RESET	Reset	
IDA_SYSREQ	Sys Request	
5250 Specific Actions		
IDA_FIELD_EXIT	Field Exit	
IDA_FIELD_PLUS	Field +	
IDA_FIELD_MINUS	Field -	
IDA_FIELD_ADVANCE	Field Advance	
IDA_FIELD_BACKSPACE	Field Backspace	
IDA_FIELD_SUB	Field SUB	
IDA_HELP	IBM Help	
IDA_ROLL_DOWN	Roll Down	
IDA_ROLL_UP	Roll Up	
IDA_ROLL_LEFT	Roll Left	
IDA_ROLL_RIGHT	Roll Right	
IDA_BACKSPACE	Backspace	
IDA_PRINT	IBM Print	
VT Actions		
IDA_ANSWERBACK	Answerback	
IDA_FIND	Find	
IDA_INSERT_HERE	Insert Here	
IDA_NEXT	Next	
IDA_PREVIOUS	Previous	

Symbolic Name	Friendly Name	Description
IDA_REMOVE	Remove	
IDA_SELECT	Select	
IDA_VT_PF1	VT PF1	Numpad PF1 key
IDA_VT_PF2	VT PF2	Numpad PF2 key
IDA_VT_PF3	VT PF3	Numpad PF3 key
IDA_VT_PF4	VT PF4	Numpad PF4 key
IDA_VT_COMMA	Numpad Comma	
IDA_NUMPAD_0	Numpad 0	
IDA_NUMPAD_1	Numpad 1	
IDA_NUMPAD_2	Numpad 2	
IDA_NUMPAD_3	Numpad 3	
IDA_NUMPAD_4	Numpad 4	
IDA_NUMPAD_5	Numpad 5	
IDA_NUMPAD_6	Numpad 6	
IDA_NUMPAD_7	Numpad 7	
IDA_NUMPAD_8	Numpad 8	
IDA_NUMPAD_9	Numpad 9	
IDA_VT_ENTER	Numpad Enter	
IDA_VT_MINUS	Numpad Minus	
IDA_VT_PERIOD	Numpad Period	
IDA_UDK_F6	UDK F6	VT User Defined Key F6
IDA_UDK_F7	UDK F7	VT User Defined Key F7
…	…	
IDA_UDK_F20	UDK F20	VT User Defined Key F20
IDA_VT_HELP	VT Help	
IDA_VT_DO	VT Do	
IDA_ADD	Add	
IDA_MULTIPLY	Multiply	
IDA_DIVIDE	Divide	
Custom VT Sequences		
IDA_VT_SAP0135	VT SAP0135	0x00 0x35
IDA_VT_CSI_M	VT CSI M	ESC [M
IDA_VT_CSI_N	VT CSI N	ESC [N
IDA_VT_CSI_O	VT CSI O	
IDA_VT_CSI_P	VT CSI P	

Symbolic Name	Friendly Name	Description
IDA_VT_CSI_Q	VT CSI Q	
IDA_VT_CSI_R	VT CSI R	
IDA_VT_CSI_S	VT CSI S	
IDA_VT_CSI_T	VT CSI T	
Windows App Keys		
IDA_APPKEY_K1	App Key 1	
IDA_APPKEY_K2	App Key 2	
…	…	
IDA_APPKEY_K16	App Key 16	
IDA_SCROLL_UPPERLEFT	Scroll Upper Left	
IDA_SCROLL_UPPERRGHT	Scroll Upper Right	
IDA_SCROLL_LOWERLEFT	Scroll Lower Left	
IDA_SCROLL_LOWERRGHT	Scroll Lower Right	
IDA_SCROLL_CENTER	Scroll Center	
IDA_SCROLL_CURSOR_CENTER	Scroll Cursor Center	
IDA_SCROLL_CURSOR_VISIBLE	Scroll Cursor Visible	
IDA_COPYALL	Copy All	Copy screen to clipboard
IDA_PASTE	Paste	Past clipboard
IDA_USTRING_1	Text 1	Send user text 1
IDA_USTRING_2	Text 2	Send user text 2
…	…	…
IDA_USTRING_64	Text 64	Send user text 64
IDA_SCRIPT_1	Script 1	Run Script 1
IDA_SCRIPT_2	Script 2	Run Script 2
…	…	…
IDA_SCRIPT_64	Script 64	Run Script 64
IDA_SIP_HIDE	SIP Hide	
IDA_SIP_SHOW	SIP Show	
IDA_SIP_TOGGLEHIDE	SIP Toggle	
IDA_SIP_LOCKDOWN	SIP Lockdown	
IDA_SIP_UNLOCK	SIP Unlock	
IDA_SIP_UP	SIP Up	
IDA_SIP_DOWN	SIP Down	
IDA_SIP_FORCEDOWN	SIP Forcedown	

Symbolic Name	Friendly Name	Description
IDA_IM_KEYBOARD	IM Keyboard	
IDA_IM_LOCKED	IM Locked	
HTML Actions		
IDA_URL_HOME	URL Home	
IDA_URL_BACK	URL Back	
IDA_URL	URL	Defines start of URL
Special Actions		
IDA_VIBRATE_100	Vibrate 100ms	
IDA_VIBRATE_200	Vibrate 200ms	
IDA_VIBRATE_500	Vibrate 500ms	
IDA_VIBRATE_1000	Vibrate 1sec	
IDA_VIBRATE_2000	Vibrate 2sec	
IDA_VIBRATE_5000	Vibrate 5sec	
IDA_BEEP_OK	Beep	
IDA_BEEP_WARN	Beep Warn	
IDA_BEEP_LOUD	Beep Loud	
IDA_KBD_ALPHA	KeyMode Alpha	
IDA_KBD_NUMERIC	KeyMode Numeric	
IDA_KBD_ALPHANUM	KeyMode AlphaNum	
IDA_KBD_UPPERALPHA	KeyMode Upper Alpha	
IDA_KBD_LOWERALPHA	Keymode Lower Alpha	
IDA_KBD_FUNCMODE	KeyMode Func	
IDA_KBD_CYCLEMODE	KeyMode Cycle	Cycle to next mode
IDA_POPUP_IPADDRESS	Show IP Address	
IDA_POPUP_MACADDRESS	Show MAC Address	
IDA_POPUP_BATTERY	Show Battery	
IDA_POPUP_TIME	Show Time	
IDA_POPUP_SERIALNUMBER	Show Serial #	
IDA_POPUP_DEVICEID	Show Device ID	
IDA_POPUP_RFINFO	Show RF info	

Appendix 2 - Properties

The properties listed in this appendix may be accessed via the GetProperty and SetProperty methods on the CETerm object. Properties marked (RO) are read-only and may not be set with SetProperty. The symbol T/F indicates a true or false value.

APPLICATION PROPERTIES

Property Name	Description
app.buildid (RO)	Program build identifier
app.name (RO)	Program name
app.script.NN	Script # NN contents, NN is 1-64
app.session.active (RO)	Currently active session
app.usertext.NN	User text # NN contents, NN is 1-64
app.version (RO)	Program version

DEVICE PROPERTIES

Property Name	Description
device.batterystatus (RO) device.battery.statustext (RO)	Current battery status string
device.battery.status (RO)	Current battery status -1 – unknown, 0 – critical, 1 – warning, 2 – low, 3 – medium, 4 – high, 5 - charging
device.battery.level (RO)	Current battery strength - 0 – 100 -1 – unknown
device.deviceid (RO)	Device ID string
device.ipaddress (RO)	IP Address of handheld
device.macaddress (RO)	MAC Address of handheld
device.platformid (RO)	Windows CE Platform ID
device.presetid (RO)	Windows CE Preset ID
device.rf.strength (RO)	RF signal strength 0-100, -2 – not associated with AP,

Property Name	Description
	-1 – unknown
device.rf.status (RO)	RF status -1 – unknown, 0 – unassociated, 1 – poor, 2 – fair, 3 – good, 4 – very good, 5 – excellent
device.serialnumber (RO)	Device serialnumber

SESSION PROPERTIES

Session properties begin with “sessionX” where X is 1 through 4. For example “session4.connection.host”. If no ‘X’ value is found, the currently active session number is used.

Property Name	Description
sessionX.connection.host	Session host (or home URL)
sessionX.connection.port	TE session port
sessionX.connection.type	Session type 3270, 5250, VT220, HTML
sessionX.printer.network.queue	Network printing queue
sessionX.printer.serial.port	Serial printing port

SCANNER PROPERTIES

Scanner properties are unique for each session. Scanner properties begin with "sessionX.scanner" where X is 1 through 4. For example "session4.scanner.enabled". If no 'X' value is found, the currently active session number is used. We use the name "scanner" for all types of barcode readers, including laser scanners and imagers. If a hardware vendor is listed, the property is specific to barcode readers made by that vendor.

NOTE: If you are changing the scanner properties for the currently activesession, you must call CETerm.PostIDA("IDA_SCAN_APPLYCONFIG", 0); for the changes to take effect.

WARNING: Not all properties are applicable to all hardware devices. Different devices may use different names to refer to the same parameters. You should look at the settings available in the CETerm configuration dialogs to determine if the property is appropriate and what values it may accept.

Property Name	Description
sessionX.scanner.enabled	Scanner is enabled. T/F
sessionX.scanner.aimerenabled	Aimer is enabled. T/F
sessionX.scanner.wedgeenabled	Allow wedge if scanner disabled in CETerm (Intermec). T/F
sessionX.scanner.focusnear	Imager focus near if true. T/F
sessionX.scanner.enhanced1d	Improved decode for poor quality barcodes
sessionX.scanner.picklistmode	Decode barcode under cross-hairs. T/F
sessionX.scanner.preamble	Barcode preamble
sessionX.scanner.postamble	Barcode postamble
sessionX.scanner.grid	Barcode grid filter (Intermec) Use OnBarcodeRead for more features.
sessionX.scanner.beamtimeout	Scan beam timeout, milliseconds
sessionX.scanner.aimertimeout	Aimer timeout, milliseconds
sessionX.scanner.aimmode	Aim mode. none, dot, slab, reticle
sessionX.scanner.redundancy	Linear security/redundancy, 0-5

COMMON SYMBOLOGY PROPERTIES

Symbology properties are unique for each session. Symbology properties begin with “sessionX.scanner.SSS” where X is 1 through 4 and SSS represents a symbology name and may be 3 or more characters long. For example “session4.scanner.upca.enabled”. If no ‘X’ value is found, the currently active session number is used. See the Symbology Names table below for SSS values.

NOTE: If you are changing the scanner properties for the currently activesession, you must call CETerm.PostIDA(“IDA_SCAN_APPLYCONFIG”, 0); for the changes to take effect.

WARNING: Not all properties are applicable to all hardware devices or all symbologies. Different devices may use different names to refer to the same parameters. You should look at the settings available in the CETerm configuration dialogs to determine if the property is appropriate and what values it may accept. This is also true for the symbologies that a device supports. You may be able to successfully change a parameter that is not supported on a device.

Last Property Level	Description
enabled	Symbology is enabled. T/F
verifycheck	Require check digit validation. T/F
redundancy	Scan redundancy flag. T/F (Symbol)
reportcheck	Report the check digit with the data. T/F
reportnumbersystem	Report UPC number system. T/F
reportcountry	Report UPC country code. T/F
reportstartstop	Report start/stop digits with barcode data. T/F
converttoupca	Convert barcode output to UPCA. T/F
converttoean13	Convert barcode output to EAN-13. T/F
supplemental2	Enable 2 digit supplemental or add-on barcode. T/F
supplemental5	Enable 5 digit supplemental or add-on barcode. T/F
supplementalrequired	Require supplemental on UPC. T/F
supplementalseparator	Insert supplemental separator. T/F
addendum	Supplemental mode. none, optional, required
minlength	Minimum barcode length. Not supported by all symbologies. See configuration dialogs for ranges.
maxlength	Maximum barcode length. Not supported by all symbologies. See configuration dialogs for ranges.
stripleading	Strip characters from start of barcode. 0-32
striptrailing	Strip characters from end of barcode. 0-32
customid	Custom symbology ID. 4 character string

CODABAR SYMBOLOGY PROPERTIES

Codabar specific symbology properties are unique for each session. Symbology properties begin with “sessionX.scanner.codabar” where X is 1 through. For example “session4.scanner.codabar.clsiediting”. If no ‘X’ value is found, the currently active session number is used.

NOTE: If you are changing the symbology properties for the currently active session, you must call “CETerm.PostIDA(“IDA_SCAN_APPLYCONFIG”, 0) for the changes to take effect.

WARNING: Not all properties are applicable to all hardware devices. Different devices may use different names to refer to the same parameters. You should look at the settings available in the CETerm configuration dialogs to determine if the property is appropriate and what values it may accept.

Last Property Level	Description
clsiediting	CLSI editing is enabled. T/F
notisediting	NOTIS editing is enabled. T/F
startstop	Start/Stop digit modes. Not all modes apply to all devices. See CETerm configuration for values on a specific device. discard, none, abcd, dc1-dc4, lowerabcd, abcd/tn*e, aa, bb, cc, dd, any

CODE39 SYMBOLOGY PROPERTIES

Code 39 specific symbology properties are unique for each session. Symbology properties begin with “sessionX.scanner.code39” where X is 1 through. For example “session4.scanner.code39.clsiediting”. If no ‘X’ value is found, the currently active session number is used.

NOTE: If you are changing the symbology properties for the currently active session, you must call “CETerm.PostIDA(“IDA_SCAN_APPLYCONFIG”, 0) for the changes to take effect.

WARNING: Not all properties are applicable to all hardware devices. Different devices may use different names to refer to the same parameters. You should look at the settings available in the CETerm configuration dialogs to determine if the property is appropriate and what values it may accept.

Last Property Level	Description
asciimode	Select ASCII mode. Not all modes apply to all devices. See CETerm configuration for values on a specific device. base, full, mixedfull
fullascii	Enable Full-ASCII mode. T/F
verifycheck39	Check digit validation mode. 0-255
reportstartstop	Report start/stop with barcode. T/F
convertocode32	Convert to Code 32 format. T/F
reportcode32prefix	Report Code 32 prefix with barcode. T/F
concatenation	Enable concatenation. T/F
stripAIAG	Remove AIAG codes. T/F
erroraccept	Allow format error. T/F

CODE 128 SYMBOLOGY PROPERTIES

Code 128 specific symbology properties are unique for each session. Symbology properties begin with “sessionX.scanner.code128” where X is 1 through. For example “session4.scanner.code128.ISBT”. If no ‘X’ value is found, the currently active session number is used.

NOTE: If you are changing the symbology properties for the currently active session, you must call “CETerm.PostIDA(“IDA_SCAN_APPLYCONFIG”, 0) for the changes to take effect.

WARNING: Not all properties are applicable to all hardware devices. Different devices may use different names to refer to the same parameters. You should look at the settings available in the CETerm configuration dialogs to determine if the property is appropriate and what values it may accept.

Last Property Level	Description
FNC1char	FNC1 character. 0-255
CIP	Enable CIP labels. T/F
ISBT	Enable ISBT 128 labels. T/F
other	Enable other 128 labels. T/F
UCCEAN	Enable UCCEAN 128 labels. T/F

UPC-EAN General Symbology Properties

UPC-EAN general symbology properties are unique for each session. Symbology properties begin with “sessionX.scanner.upc-ean” where X is 1 through. For example “session4.scanner.upc-ean.bookland”. If no ‘X’ value is found, the currently active session number is used.

NOTE: If you are changing the symbology properties for the currently active session, you must call “CETerm.PostIDA(“IDA_SCAN_APPLYCONFIG”, 0) for the changes to take effect.

WARNING: Not all properties are applicable to all hardware devices. Different devices may use different names to refer to the same parameters. You should look at the settings available in the CETerm configuration dialogs to determine if the property is appropriate and what values it may accept.

Last Property Level	Description
bookland	Enable Bookland labels. T/F
coupon	Enable Coupon labels. T/F
lineardecode	Enable linear decode. T/F
supplemental2	Enable 2 digit supplemental or add-on barcode. T/F
supplemental5	Enable 5 digit supplemental or add-on barcode. T/F
supplementalretry	Supplemental decode retry count. 2-10
randomweightcheckdigit	Enable random weight check digit. T/F
supplementalmode	Supplemental mode. none, always, auto
securitylevel	Decode security level. none, all, ambiguous

SYMBOLOGY NAMES

Symbology properties begin with “sessionX.scanner.SSS” where X is 1 through 4 and SSS represents a symbology name and may be 3 or more characters long. The following table lists all available symbology names.

WARNING: Not all symbologies are applicable to all hardware. Different devices may use different names to refer to similar symbologies, e.g., upce and upce0. You should look at the symbologies available in the CETerm configuration dialogs to determine the correct name.

Symbology Name	Description
ames	Ames
auspostal	Australian Postal
aztec	Aztec
bpo	British Postal
canpostal	Canadian Postal
chinapostal	China Postal
codabar	Codabar
codablock	Codablock
code11	Code 11
code16k	Code 16k
code32	Code 32
code39	Code 39
code49	Code 49
code93	Code 93
code128	Code 128
composite	Composite AB and C
couponcode	Coupon Code
d2of5	Discrete (standard) 2 of 5
datamatrix	Datamatrix
delta	Delta Code
dutchpostal	Dutch Postal
ean8	EAN-8
ean13	EAN-13
i2of5	Interleaved 2 of 5
iata25	IATA 2 of 5
idtag	ID Tag
isbt	ISBT
japanpostal	Japan Postal
koreapostal	Korea Postal
label45	Label 45
m2of5	Matrix 2 of 5

Symbology Name	Description
maxicode	Maxicode
mesa	Mesa
micropdf	Micro PDF
msi	MSI
pdf	PDF 417
pdf417	PDF 417
pharma39	Pharma 39
planet	Planet
plessey	Plessey
posicode	Posicode
postnet	Postnet
qrcode	QR Code
rss	RSS 14
rss14	RSS 14
rssexp	RSS Expanded
rsslim	RSS Limited
rssltd	RSS Limited
telepen	Telepen
tlc39	TLC 39
trioptic39	Trioptic 39
ukpostal	British (UK) Postal
upca	UPC-A
upce	UPC-E
upce0	UPC-E0
upce1	UPC-E1
upc-ean	UPC-EAN General Settings
usplanet	US Planet
uspostnet	US Postnet
usps4cb	USPS 4CB

Appendix 3 – Symbology LabelTypes

This appendix contains a list of symbology labeltypes that are returned in the "type" argument of OnBarcodeRead. These are also available to a ScannerNavigate META tag handler. Please note that not all hardware devices return these values. You may need to test scan a known barcode to find the labeltype value for that barcode.

LabelType Character	Hexadecimal Value	Symbology
#	0x23	Plessey
&	0x24	Telepen
%	0x25	Codablock A
$	0x26	Codablock F
‘ (single quote)	0x27	Matrix 2 of 5
(	0x28	Code 49
)	0x29	Code 16K
*	0x2A	Ankercode
+	0x2B	Aztec
, (comma)	0x2C	Korea Postal
0	0x30	UPC-E or UPC-E0
1	0x31	UPC-E1
2	0x32	UPC-A
3	0x33	MSI
4	0x34	EAN-8
5	0x35	EAN-13
6	0x36	Codabar
7	0x37	Code 39
8	0x38	Discrete 2 of 5
9	0x39	Interleaved 2 of 5
: (colon)	0x3A	Code 11
; (semi-colon)	0x3B	Code 93
<	0x3C	Code 128
>	0x3E	IATA 2 of 5
?	0x3F	EAN 128

LabelType Character	Hexadecimal Value	Symbology
@	0x40	PDF 417
A	0x41	ISBT 128
B	0x42	Trioptic 39
C	0x43	Coupon Code
D	0x44	Bookland
E	0x45	Micro PDF
F	0x46	Code 32
G	0x47	Macro PDF
H	0x48	Maxicode
I	0x49	Datamatrix
J	0x4A	QR Code
K	0x4B	Macro Micro PDF
L	0x4C	RSS-14
M	0x4D	RSS Limited
N	0x4E	RSS Expanded
V	0x56	Composite AB
W	0x57	Composite C
X	0x58	TLC 39
a	0x61	US Postnet
b	0x62	US Planet
c	0x63	UK (British) Postal
d	0x64	Japan Postal
e	0x65	Australian Postal
f	0x66	Dutch Postal
g	0x67	Canadian Postal
p	0x70	Mesa
q	0x71	OCR
r	0x72	China Postal
s	0x73	Posicode
t	0x74	USPS4CB
u	0x75	ID Tag

Appendix 4 - Constants

This appendix contains various constants that are used by CETerm Automation Objects. Many of these constants are a direct representation of the equivalent values from the Windows CE system APIs and constants. These constants are presented as JavaScript variables for direct inclusion in scripts.

MESSAGEBOX CONSTANTS

```
// MessageBox flags
// See Microsoft SDK for documentation.
var MESSAGEBOX_FLAG_OK                = 0x00000000;
var MESSAGEBOX_FLAG_OKCANCEL          = 0x00000001;
var MESSAGEBOX_FLAG_ABORTRETRYIGNORE = 0x00000002;
var MESSAGEBOX_FLAG_YESNOCANCEL       = 0x00000003;
var MESSAGEBOX_FLAG_YESNO             = 0x00000004;
var MESSAGEBOX_FLAG_RETRYCANCEL       = 0x00000005;

var MESSAGEBOX_FLAG_ICONERROR        = 0x00000010;
var MESSAGEBOX_FLAG_ICONQUESTION     = 0x00000020;
var MESSAGEBOX_FLAG_ICONWARNING      = 0x00000030;
var MESSAGEBOX_FLAG_ICONINFORMATION = 0x00000040;

var MESSAGEBOX_FLAG_DEFBUTTON1 = 0x00000000;
var MESSAGEBOX_FLAG_DEFBUTTON2 = 0x00000100;
var MESSAGEBOX_FLAG_DEFBUTTON3 = 0x00000200;
var MESSAGEBOX_FLAG_DEFBUTTON4 = 0x00000300;

var MESSAGEBOX_FLAG_APPLMODAL     = 0x00000000;
var MESSAGEBOX_FLAG_SETFOREGROUND = 0x00010000;
var MESSAGEBOX_FLAG_TOPMOST       = 0x00040000;

// MessageBox returned values
var MESSAGEBOX_IDOK     = 1;
var MESSAGEBOX_IDCANCEL = 2;
var MESSAGEBOX_IDABORT  = 3;
var MESSAGEBOX_IDRETRY  = 4;
var MESSAGEBOX_IDIGNORE = 5;
var MESSAGEBOX_IDYES    = 6;
var MESSAGEBOX_IDNO     = 7;
```

PLAYSOUND CONSTANTS

```
// PlaySound flags
// See Microsoft SDK for documentation.
var PLAYSOUND_FLAG_ASYNC = 0x00000001;  // Play asynchronously
var PLAYSOUND_FLAG_NODEFAULT = 0x00000002;  // No default sound
```

```
var PLAYSOUND_FLAG_LOOP = 0x00000008;  // Repeat play, needs ASYNC.
var PLAYSOUND_FLAG_NOSTOP = 0x00000010;  // Don't stop current sound
var PLAYSOUND_FLAG_NOWAIT = 0x00002000;  // Don't play if driver busy
```

FILE ATTRIBUTE CONSTANTS

```
// File attirbute flags
// See Microsoft SDK for documentation.
var FILE_ATTRIBUTE_READONLY     = 0x00000001;
var FILE_ATTRIBUTE_HIDDEN       = 0x00000002;
var FILE_ATTRIBUTE_SYSTEM       = 0x00000004;
var FILE_ATTRIBUTE_DIRECTORY    = 0x00000010;
var FILE_ATTRIBUTE_ARCHIVE      = 0x00000020;
var FILE_ATTRIBUTE_INROM        = 0x00000040;
var FILE_ATTRIBUTE_ENCRYPTED    = 0x00000040;
var FILE_ATTRIBUTE_NORMAL       = 0x00000080;
var FILE_ATTRIBUTE_TEMPORARY    = 0x00000100;
var FILE_ATTRIBUTE_COMPRESSED   = 0x00000800;
var FILE_ATTRIBUTE_ROMSTATICREF = 0x00001000;
var FILE_ATTRIBUTE_ROMMODULE    = 0x00002000;
```

REGISTRY CONSTANTS

```
// Registry constants
// See Microsoft SDK for documentation.
// Root key names
var HKEY_CLASSES_ROOT  = "HKEY_CLASSES_ROOT";
var HKEY_CURRENT_USER  = "HKEY_CURRENT_USER";
var HKEY_LOCAL_MACHINE = "HKEY_LOCAL_MACHINE";
var HKEY_USERS         = "HKEY_USERS";

// Data types
var REG_SZ             = "REG_SZ";
var REG_DWORD          = "REG_DWORD";
var REG_BINARY         = "REG_BINARY";
var REG_MULTI_SZ       = "REG_MULTI_SZ";
var REG_EXPAND_SZ      = "REG_EXPAND_SZ";

// Returned Status
var REGISTRY_SUCCESS         = 0;
var REGISTRY_FAIL            = -1;
var REGISTRY_BAD_HIVE        = -2;
```

```
var REGISTRY_BAD_KEYNAME     = -3;
var REGISTRY_BAD_DATATYPE    = -4;
var REGISTRY_BAD_VALUE       = -5;
var REGISTRY_BAD_VALUEFORMAT = -6;
var REGISTRY_OUTOFMEMORY     = -7;
```

BROWSER ERROR CONSTANTS

```
// Navigate Error HRESULT status codes
// See Microsoft SDK for documentation.
//
// URL string is not valid.
var INET_E_INVALID_URL               = 0x800C0002;
// No session found.
var INET_E_NO_SESSION                = 0x800C0003;
// Unable to connect to server.
var INET_E_CANNOT_CONNECT            = 0x800C0004;
// Requested resource is not found.
var INET_E_RESOURCE_NOT_FOUND        = 0x800C0005;
// Requested object is not found.
var INET_E_OBJECT_NOT_FOUND          = 0x800C0006;
// Requested data is not available.
var INET_E_DATA_NOT_AVAILABLE        = 0x800C0007;
// Failure occurred during download.
var INET_E_DOWNLOAD_FAILURE          = 0x800C0008;
// Authentication required.
var INET_E_AUTHENTICATION_REQUIRED   = 0x800C0009;
// Required media not available or valid.
var INET_E_NO_VALID_MEDIA            = 0x800C000A;
// Connection timed out.
var INET_E_CONNECTION_TIMEOUT        = 0x800C000B;
// Request is invalid.
var INET_E_INVALID_REQUEST           = 0x800C000C;
// Protocol is not recognized.
var INET_E_UNKNOWN_PROTOCOL          = 0x800C000D;
// Failed due to security issue.
var INET_E_SECURITY_PROBLEM          = 0x800C000E;
// Unable to load data from the server.
var INET_E_CANNOT_LOAD_DATA          = 0x800C000F;
// Unable to create an instance of the object.
var INET_E_CANNOT_INSTANTIATE_OBJECT = 0x800C0010;
// Attempt to redirect the navigation failed.
var INET_E_REDIRECT_FAILED           = 0x800C0014;
// Navigation redirected to a directory.
var INET_E_REDIRECT_TO_DIR           = 0x800C0015;
// Unable to lock request with the server.
var INET_E_CANNOT_LOCK_REQUEST       = 0x800C0016;
// Reissue request with extended binding.
var INET_E_USE_EXTEND_BINDING        = 0x800C0017;
// Binding is terminated.
```

```
var INET_E_TERMINATED_BIND           = 0x800C0018;
// Permission to download is declined.
var INET_E_CODE_DOWNLOAD_DECLINED    = 0x800C0100;
// Result is dispatched.
var INET_E_RESULT_DISPATCHED         = 0x800C0200;
// Cannot replace a protected SFP file.
var INET_E_CANNOT_REPLACE_SFP_FILE   = 0x800C0300;
```

Glossary

Automation Objects
Objects internal to CETerm that provide access to device, application, and session features from the script engine.

CEBrowseX
A Naurtech ActiveX control which provides access to the CETerm Automation Objects from a Windows Mobile device.

external
This is the name of an internal object in the DOM of the Windows CE 5.0 browser that gives access to the CETerm Automation Objects.

IDA Action Code
An IDA Action Code defines a special device, application, or emulation action within the Naurtech Smart Clients. IDA codes can be tied to keys, or KeyBars, and invoked via META tags or JavaScript. See the Appendix for a list of values.

Index

A

AbortScript · 22, 23
AddMetaItem · 28, 29, 42, 49
Alert · 32

B

Beep · 32
Browser object · 28

C

CETerm object · 22, 24, 26, 75
ClearAllTimers · 22, 23
ClearInterval · 22, 23, 25
ClearTimeout · 22, 23, 26, 51, 56
Copy · 34
CreateDirectory · 34, 35

D

Delete · 35
DeleteKey · 37, 38
DeleteValue · 37, 38

E

EnumerateKeys · 37, 38
EnumerateValues · 37, 38
Exec · 31, 32, 33, 34, 59
ExpectMonitor · 18, 25, 26, 45, 52, 53, 54, 55, 56, 57, 59

F

File object · 33, 34, 36, 37
FlushKey · 37, 38

G

GetAttributes · 34, 35, 36, 61
GetErrorMessage · 31, 32
GetInput · 31, 59
GetOpenFileName · 34, 35
GetProperty · 22, 23, 41, 43, 47, 51, 75
GetSaveFileName · 34, 35
GetText · 27, 28, 54
GetTextLine · 20, 27, 48, 49
GetTextRect · 27, 28
GetValueType · 37, 38

I

IDA Action Codes · 15, 66

K

KillProcess · 31, 32

L

Load at Startup · 14, 63

M

MessageBox · 32, 65, 87
Move · 36

N

Navigate · 29

O

OnBarcodeRead · 10, 15, 16, 17, 18, 40, 41, 49, 77, 85
OnDocumentDone · 40, 41, 42, 60
OnNavigateError · 40, 42, 43, 44, 45
OnNetCheckFailed · 40, 44, 45
OnSessionConnect · 18, 40, 45
OnSessionDisconnect · 40, 46
OnSessionDisconnected · 40, 47
OnSessionReceive · 40, 47, 48
OnSessionSwitch · 40, 48, 49
OnStylusDown · 20, 40, 49
OnWakeup · 40, 50
OS object · 22, 23, 31, 33, 34

P

PlaySound · 20, 22, 23, 32, 33, 51, 52, 87
PlayTone · 22, 23, 24, 32, 33, 63
PostIDA · 24

R

Read · 36
ReadValue · 37, 38, 39, 62
ReadValueVBArray · 37, 39
Registry object · 34, 37, 39
RemoveDirectory · 34, 36
RunScript · 22, 24, 28, 29, 42

S

Screen object · 27, 28
SendIDA · 17, 19, 21, 22, 24, 46
SendText · 16, 17, 19, 22, 24, 25, 41, 54
Session object · 25, 26, 27
SetAttributes · 34, 36
SetInterval · 22, 25
SetProperty · 25, 75
SetScriptTimeout · 26
SetTimeout · 22, 26, 51, 56, 59, 60
Sleep · 32, 33, 58, 69, 70
StopSound · 32, 33

T

TextInput · 26, 30, 31, 58

W

WaitForProcess · 32, 33
Write · 36
WriteValue · 37, 39, 61