



# Web Browser Programming Guide

for Version 5.7 or later

Naurtech Industrial Web Browser  
And  
Terminal Emulation Clients

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 to make changes to it from time to time without any obligation to notify any person or organization of such revision or changes.

## Trademarks

CETerm<sup>®</sup>, CE3270<sup>™</sup>, CE5250<sup>™</sup>, CEVT220<sup>™</sup> 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 5.7.0 or later of Naurtech Industrial Web Browser and Terminal Emulation clients.**

## Table of Contents

Copyright Notice .....	2
Trademarks.....	2
Software Version .....	2
Table of Contents .....	3
Preface .....	5
Assumptions .....	5
Conventions used in this Manual.....	5
Additional Documentation.....	6
Online Support Wiki.....	6
1.0 Introduction.....	7
1.1 Feature Highlights.....	8
1.2 Browser Differences For Windows CE Platforms .....	10
2.0 Common Tasks .....	11
2.1 Scanner Input.....	11
2.2 Key Actions .....	12
2.3 Text Input Elements .....	13
2.4 IDA Action Codes.....	13
2.5 Device Control From JavaScript .....	14
2.6 Device Properties and CETerm Configuration.....	16
3.0 Special HTML META Tags .....	18
3.1 Application.....	21
3.2 Battery .....	21
3.3 BatteryNavigate .....	22
3.4 Command.....	24
3.5 CursorPos .....	24
3.6 ErrorNavigate .....	25
3.7 GetUnitInformation.....	26
3.8 HomeKey .....	27
3.9 IDA .....	27
3.10 MoveSIP.....	28
3.11 OnAllKeys .....	29
3.12 OnKey .....	30
3.13 PowerOn .....	32
3.14 Reboot.....	32
3.15 Scanner.....	33
3.16 ScannerNavigate .....	34
3.17 SetDate .....	36
3.18 SetTime.....	36
3.19 Signal .....	37
3.20 SignalNavigate .....	38
3.21 SIP .....	39
3.22 SIPUp.....	40
3.23 TextSize .....	40
3.24 TimerInterval .....	41
3.25 TimerNavigate.....	41
3.26 ZebraLabel_Complete or PLSeriesLabel_Complete .....	42
3.27 ZebraLabel_Print or PLSeriesLabel_Print .....	43

4.0 Advanced Topics .....	45
4.1 Navigating to Pre-configured URLs .....	45
4.2 Controlling the Scanner .....	45
4.3 Input Focus and the Tab Key .....	47
4.3.1 Windows CE (IE6CE) Example .....	48
4.3.2 Windows Mobile (PIE) Example .....	50
4.4 Session Launcher .....	52
4.5 How to Identify the Current Browser .....	54
4.6 Device Information .....	55
4.7 Symbol Web Client .....	55
5.0 Printing from HTML .....	57
5.1 Printing with a META Tag .....	57
5.2 PrintString and Print Methods .....	57
5.3 NAURTECH:PRINT Tag .....	58
5.4 Direct SerialPort Printing.....	58
5.5 ActiveX Printing Controls .....	58
6.0 CEBrowseX Control.....	59
Syntax .....	59
ClassID.....	60
Methods .....	60
Status = PrintString( printData ) .....	60
Status = Print( printData).....	60
Properties.....	60
EventHandlers.....	61
Appendix 1 - Virtual Key Codes.....	62
Glossary.....	66
Index .....	68

## 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 working knowledge of:

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


## Conventions used in this Manual

This manual uses the following typographical conventions:

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

- Precautionary notes or tips are presented as follows

**TIP:** Text associated with a specific tip

-  represents new version specific information
- Text associated with samples is presented as follows. We use lower case in most samples for readability.

```
<html>
<head>
<title>Naurtech Web Browser</title>
<meta http-equiv="Scanner" content="Enabled">
```

```
<meta http-equiv="ScannerNavigate"
      Content="Javascript:onscan('%s','%s','%s','%s','%s');">
</head>
...
</html>
```

## Additional Documentation

The Naurtech Industrial Web Browser is an integral feature of Naurtech Terminal Emulation Clients: CETerm, CE5250, CE3270 and CEVT220. Please refer to the CETerm User's Manual for installation and configuration information. The User's Manual may be downloaded from the "Support" section of [www.naurtech.com](http://www.naurtech.com). You will also need the CETerm Scripting Guide which contains Appendices describing IDA codes, CETerm property names, and Windows Virtual Key (VK) codes.

## Online Support Wiki

Although we strive to keep this manual up to date, you may find our online Support Wiki useful for the latest features, sample HTML, and troubleshooting tips. You can access the Support Wiki from our website at:

[www.naurtech.com](http://www.naurtech.com) → Support → Support Wiki

## 1.0 Introduction

The Naurtech Industrial Web Browser provides a robust and flexible environment for Web based applications which are accessed from an industrial mobile device. This browser is available for most Windows CE platforms; including Windows CE .NET (4.2), Windows CE 5.0, Windows CE 6.0, Windows Mobile 2003, Windows Mobile 5.0 and Windows Mobile 6.0 based devices.

Device tailored versions of the browser are available for most industrial terminals. These versions integrate with the peripherals on each device, such as barcode scanner, magnetic stripe reader, RFID reader and Bluetooth printer. The Naurtech Web Browser provides control of the peripherals and simplifies actions such as data collection, validation, and printing.

All Naurtech Web Browsers are integrated with one or more Terminal Emulations (TE) which allows a natural migration path from legacy text-based TE applications to newer Web applications. The Web applications can be presented in a familiar, single-purpose (locked down) configuration which uses keys, the touch screen, or both for user interactions.

The Naurtech Web Browser offers control of the device peripherals and settings via JavaScript extensions, ActiveX controls, and special HTML META tags. This guide is written primarily to describe these extensions and custom features. Please consult the standard references for details on JavaScript, HTML syntax, the browser Document Object Model (DOM), and other aspects of Dynamic HTML. Please refer to the CETerm User's Manual for details on basic usage and configuration of the Naurtech Web Browser clients.

There is no current standard for the browser extensions and META tags that have been added to industrial Web browsers. Within the Naurtech Browser, we strive to support all the ad-hoc and de-facto extensions available in other products. In many cases, the Naurtech Web Browser is a "drop-in" replacement for these other products. In addition, we support nearly uniform behavior across a wide spectrum of devices from every major hardware device manufacturer.

Beyond these basic extensions, we have added many unique features to enable you to build more powerful business applications.

## 1.1 FEATURE HIGHLIGHTS

Following are some of the special features in the Naurtech Web Browser.

- **Access Control / Device Lockdown.** Access controls allow administrators to hide the start bar and to prevent users from exiting the Naurtech Browser. You can hide just the Windows CE “Start” button, the whole Start bar and/or the application menu bars and toolbars. Users are prevented from navigating to un-authorized Web sites as is possible with Pocket Internet Explorer.
- **Multiple Browser Sessions.** All Naurtech clients allow up to 5 simultaneous sessions. Each session can be connected to a different Web application. This allows quick access to separate applications. Each session supports a unique scanner configuration.
- **Network Awareness.** The Naurtech Web Browser can be configured to verify the network path to the host prior to navigation. Further, the loss of the network, or other failures during navigation can be detected and error recovery applied. These features contribute to robust Web applications which don’t require the user to re-start when faced with network disruptions.
- **Enhanced native HTML text INPUT.** Under Windows Mobile, the standard HTML text INPUT element will popup the SIP when it receives focus, and will not respond to the Tab key to advance the focus. The Naurtech Web Browser can prevent the SIP popup and enables Tab based navigation, without resorting to ActiveX input objects. Native HTML text INPUT elements are easier to use and help maintain consistency across different platforms.
- **Scanner Control via JavaScript.** Scanner input can be intercepted by JavaScript methods for data validation and editing. The scanner can also be enabled or disabled with META tags or from JavaScript.
- **Integration with CTerm Scripting and Automation Objects.** The web browser has access to the independent JavaScript engine running within CTerm and the associated Automation Objects. This independent engine can enhance legacy Web pages and add functionality. The Automation Objects provide access to more features in CTerm and the underlying Windows OS. Please refer to the CTerm Scripting Guide for full details.
- **Key Driven Interaction.** Hardware keys can be re-assigned with HTML META tags to activate any desired action. Keys can be used to navigate to specific pages, to clear fields, submit forms, and even switch to different sessions.

- **On Screen Indicators.** Battery and WLAN (RF) strength can be displayed with on-screen indicators. The meters may also be displayed within KeyBar buttons. On some devices, keypad state indicators can also be displayed on-screen.
- **Printing from HTML.** Several techniques are available to send print content from an HTML page to a printer. The printer may be accessed via a serial port, IrDA, Bluetooth or the WLAN.
- **Notification Features.** Sound (.wav) files, tone generators (beepers), and vibrators can all be activated on devices which have these capabilities
- **Context Menus.** Custom context menus can be defined to provide access to special actions without tying up valuable screen real-estate.

## **1.2 BROWSER DIFFERENCES FOR WINDOWS CE PLATFORMS**

The Naurtech Web Browser uses a native Microsoft Windows CE browser component for navigation, HTML parsing and rendering. Naurtech wraps this component with a rich feature set to provide the robust and flexible industrial browser.

There are two families of Windows CE platforms. Windows Mobile is built atop the Windows CE OS but we refer to it as the “Windows Mobile” platform. If a device does not use “Windows Mobile”, it is said to use the “Windows CE” platform (e.g., Windows CE 5.0, Windows CE 6.0).

Different Microsoft browser components are available on different platforms and have different capabilities. In general, Windows Mobile platforms provide “Pocket Internet Explorer” (PIE) and Windows CE (4.2, 5.0, 6.0) platforms provide “Internet Explorer 6 for Windows CE” (IE6CE). The PIE browser was optimized for a small memory footprint and is missing many standard behaviors (e.g., onkeydown events) that are expected in contemporary browsers. The IE6CE browser is similar to IE 6 on a desktop PC and has a more complete set of behaviors, but does not have all the features you would find in a contemporary desktop browser.

Naurtech overcomes many of the limitations of PIE using META tags and other enhancements, but Web application designers must review and understand the limitations and tailor their applications for the target browser. In general, we recommend using the Windows CE platform when practical and when the most complete set of browser features (IE6CE) is required.

Use the following link to find out more about the browser differences, or if it does not resolve, try searching for “MSDN Choosing an Internet Browser”:

<http://msdn.microsoft.com/en-us/library/aa451916.aspx>

The PIE browser has also been called the “Mobile Internet Browser” by Microsoft.

The newer Windows Mobile (6.1.4 and later) platforms contains a newer browser called “Internet Explorer Mobile 6” (IEM6). This browser is similar to IE6CE but is not yet available for hosting. Because the Naurtech Web Browser uses PIE on Windows Mobile devices, you will observe differences in behavior between the Naurtech Web Browser and the native IE on devices with IEM6. Using PIE also means that the Naurtech Browser will behave the same on Windows Mobile 5 and Windows Mobile 6 devices.

## 2.0 Common Tasks

This section describes some common ways that enhanced features can be used within a Web based application. Here we show how to manage scanner input, invoke actions via keys, and interact with the handheld device. Only small code “snippets” are shown. For complete details see the reference sections of this manual. These tasks help to illustrate the power of the Naurtech Web Browser for building Web based applications.

### 2.1 SCANNER INPUT

The barcode scanner, and other readers such as magnetic card readers and RFID readers are typically integrated with the Naurtech Browser. The configuration of the reader is maintained by the Naurtech Browser and is managed to allow an independent configuration for each host session. When data is available from a reader, it is directed to the current host session. The data is typically inserted at the current cursor (or focus) location. For a Web page with multiple text input elements, this can be problematic. The focus may be in the wrong text element, or if the focus is not in a text element, the input may be lost.

A better way to receive scanner input is to use an extension, which directs the data to a JavaScript method or submits it to a URL. To enable this action, you must define a special HTML META tag and a JavaScript method to process the data. Here is a typical META tag:

```
<meta http-equiv="ScannerNavigate"
content="Javascript:onscan('%s','%s','%s','%s','%s');">
```

When the scanner reads a barcode, each of the ‘%s’ items will be filled with information from the scan and the onscan method will be invoked. Here is a typical method which shows what each argument contains

```
<script language=javascript>
function onscan(data, source, type, time, length)
{
  alert("The barcode scanned was " + data +
        "\nThe symbology was " + type +
        "\nScanned at " + time +
        "\nWith a length of " + length);
}
</script>
```

This method simply presents a popup message with the scanner data. More typically, the data would be validated and inserted into a text element. The HTML form containing that element may also be submitted:

```
<script language=javascript>
function onscan(data, source, type, time, length)
{
  if (length > 5)
  {
    document.form[0].barcode.value = data;
    document.form[0].submit();
  }
}
<\script>
```

## **2.2 KEY ACTIONS**

In many situations, a Web application will be written to make special use of the keys on a handheld. The device may be used without a stylus or there may be function keys (Fx) which the application uses to perform special actions. The Naurtech Web Browser has several special features to make use of the hardware keys. These features are especially important on Windows Mobile based systems. The Windows Mobile PIE browser does not have native support for special key actions, but the Naurtech Web Browser overcomes this limitation.

The simplest way to assign a key to a special action is to use a special HTML META tag. This will instruct the browser to execute a JavaScript method or navigate to a pre-specified URL when the key is pressed:

```
<meta http-equiv="OnKeyVK_F1" content="Javascript:onF1key();">
...
<script language=javascript>
function onF1key()
{
  // Clear entry
  document.form[0].barcode.value = "";
}
<\script>
```

or

```
<meta http-equiv="OnKey0x70" content="home.htm"><!--F1-->
```

It is important to know that the OnKey META tag acts as a “hotkey” and it will activate the action even if the focus is in a text input element. If you assign an action to a normal key such as ‘1’ then you will be unable to enter the ‘1’ as a character anywhere on the page. Techniques are available to ignore the hotkey action within text input elements.

## **2.3 TEXT INPUT ELEMENTS**

All Web applications will use text input elements at some point to collect information such as a barcode or count. The standard text input element is the HTML INPUT such as

```
<input type="text" name="count" maxlength="10" size="10">
```

The capabilities of the text input element are different for different Windows CE platforms (see Section 1.2). The Windows Mobile PIE browser has the most limited text input. For example, this input element does not support special event handlers such as “OnKeyPress”, does not respond to the Tab key to advance the focus, and will popup the Soft Input Panel (SIP) whenever focus is received by the element.

The Naurtech Web Browser corrects these PIE deficiencies by adding Tab key support, locking down the SIP when desired, and using the OnKey META tag for special actions.

On Windows CE platforms (IE6CE browser), the native text INPUT element fully supports the OnKeyPress handler, as well as other events, and will behave like a desktop browser.

## **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 Industrial Browser. IDA Action Codes can, for example, 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 the CETerm Scripting Guide. Almost any action which can be invoked by a KeyBar or assigned to a hardware key, can be invoked by an IDA code. Under the Web Browser, IDA codes can be sent to the program in several different ways. They can be in a special META tag, in an HTML link, or sent via JavaScript.

Here is a sample which pops up the SIP when a page loads:

```
<meta http-equiv="IDA" content="IDA_SIP_SHOW">
```

Or, you can toggle the SIP visibility from a link:

```
<a href="ida:IDA_SIP_TOGGLEHIDE">Toggle Soft Input Panel (SIP)</a>
```

Or, you can perform the action from JavaScript by setting the document location:

```
<script language=javascript>
function togglehide()
{
    // Toggle the SIP visibility
    // This format may not work for Windows Mobile (PIE)
    location.href = "ida:IDA_SIP_TOGGLEHIDE";

    // Or (remove the comment characters)
    // document.location = "ida:IDA_SIP_TOGGLEHIDE";

    // Or (remove the comment characters)
    // window.navigate("ida:IDA_SIP_TOGGLEHIDE" );
}
<\script>
```

The next section describes additional methods for invoking IDA Action Codes from within a JavaScript method.

## **2.5 DEVICE CONTROL FROM JAVASCRIPT**

The Naurtech Web Browser is tailored to the features of most handheld devices. If the handheld has a vibrator or tone generator, we provide access to those features through JavaScript. We also provide access to most operations of the Browser client, such as switching to other sessions, or retrieving device or configuration information.

To access these features you use “CETerm Automation Objects”. These objects are described in the CETerm Scripting Guide. The technique used to gain access to the Automation Objects depends on the Windows CE platform. As described in Section 1.2, different Windows CE platforms support different browsers. With the Windows Mobile platform (PIE browser) an ActiveX control is needed, but on Windows CE platforms (IE6CE browser) a predefined “external” object is used.

Following is a technique to gain access to CETerm Automation Objects that will work on both Windows CE and Windows Mobile platforms. The technique results in a JavaScript variable “ext” which holds an object reference.

```
<script language=javascript>
// Put this <script> element in the <head> of the page.
// Resolve ext reference one time, when page loads.
// WARNING: Make sure ext is not already used by your scripts.
// Examples which reference top-level objects:
// ext.CETerm.PostIDA( "IDA_SIP_SHOW", 0 ); // show SIP
// ext.OS.File.Append( "\\myfile.txt", "content" );
```

```
// var sp1 = ext.Device.SerialPort(1); // Get a SerialPort object
//
var ext = null; // global variable, not declared in a function
if (typeof external === "object")
{
    // Windows CE
    // external is already defined in global namespace
    ext = external;
}
else
{
    // Windows Mobile
    // Create CEBrowseX for top-level object access
    ext = new ActiveXObject( "Cebrowsex.IdaCtl" );
}
</script>
```

Once the “ext” value is declared and set, it can be used to access any of the top-level CETerm Automation Objects, “CETerm”, “Device”, and “OS”.

**NOTE:** Previously an <object> tag was often used to create the CEBrowseX control. We recommend using the “new ActiveXObject()” technique, but the <object> tag can still be used and is described in Chapter 6.

The following example shows how to sound a tone with an IDA command.

```
<html>
<head>
<meta http-equiv="OnKey0x31" content="javascript:mybeep();">
<script>
    ...Above script to define and set ext
</script>
</head>
<body>
<a href="Javascript:mybeep();">Tap me or press 1 for beep</a><br>
...
<script language=javascript>
function mybeep()
{
    // Use the ext value defined above
    ext.CETerm.PostIDA( "IDA_BEEP_LOUD", 0 );
}
</script>
</body>
</html>
```

The PostIDA method sends an IDA Action Code as described in the previous section. For example, action codes can activate the vibrator (for 500 millisecond) (IDA\_VIBRATE\_500), switch to a different CETerm session (IDA\_SESSION\_S1) and many more actions.

## **2.6 DEVICE PROPERTIES AND CETERM CONFIGURATION**

The CETerm object can also be used to access device properties and to read or set portions of the CETerm configuration. The following sample shows how this can be used

```
...
<script>
  ...Above script to define and set ext
</script>
...
<body onload="javascript:fetchvalues();" >
<form id="form1" name="form1">
Property Features<br>
<input type="text" id="serialnumber" name="serialnumber"
size="20"><br>
<input type="text" id="ipaddress" name="ipaddress" size="20"><br>
</form>
...
<script language=javascript>
function fetchvalues()
{
  document.form1.serialnumber.value =
    ext.CETerm.GetProperty( "device.serialnumber" );
  document.form1.ipaddress.value =
    ext.CETerm.GetProperty( "device.ipaddress" );
}
</script>
...
```

The available properties are documented in the CETerm Scripting Guide. One useful property gives access to the “User Text” area of CETerm. User Text properties are strings of characters which may contain IDA codes and which can be “sent” to an emulator. User Text strings are often tied to hardware keys to simplify text entry or to create “mini-macros” of IDA actions. From the browser, the User Text can be used as a general device-local persistent storage. They can also be used to send the user to special pre-configured URL’s.

Here is an example of using a User Text area for persistent storage:

```
...
<script>
  ...Above script to define and set ext
```

```
</script>
...
<body onload="javascript:loadfields();">
<form id="form1" name="form1">
Login Page<br>
<input type="text" id="user" name="user" size="20"><br>
<input type="password" id="password" name="password" size="20"><br>
</form>
...
<script language=javascript>
function loadfields()
{
    var username = ext.CETerm.GetProperty( "app.usertext.1" );
    if (username)
    {
        // Load with cached user name for this device
        document.form1.user.value = username;
    }
}
...
</script>
```

When the login is successful the application would store the current user name for the next login attempt

```
...
function onlogin( username )
{
    // Login was successful
    ext.CETerm.SetProperty( "app.usertext.1", username );
}
...
```

There are some limitations with User Text. The values are shared among all emulator sessions and currently there are only 64 slots available.

### 3.0 Special HTML META Tags

This section describes the special META tags (or elements) that are recognized by the Naurtech Web Browser and are used to convey special instructions to the browser. These special META tags can assign hotkeys, control the scanner, configure the device, and perform other tasks. The META tags use the standard HTML format but are not recognized by standard browsers.

A META tag has the form

```
<meta http-equiv="identifier" content="valueString">
```

The Naurtech Browser supports a unique set of “identifier” values. However, because there is no current standard for these META tags, we also recognize most of the ad-hoc and de-facto extensions available in other products. In particular, we support most of the Symbol Pocket Browser and Intermec iBrowse META tag identifiers. For the iBrowse META tags, we recognize the identifiers with or without the leading “iBrowse\_” or “IB\_” text.

**TIP:** In general, we are not adding new functionality as META tags. The CETerm Automation Objects, accessed from JavaScript, provide a more reliable and robust environment. META tags can only be processed when a new page is loaded. Automation Objects can be used at any time with JavaScript and AJAX techniques. The CETerm automation object can also simulate the legacy META tags to provide more flexible behavior and improve legacy Web applications.

**TIP:** We recommend using standard browser handlers, such as, onload and setTimeout(), together with CETerm Automation Objects rather than META tags where possible. The META tags “ScannerNavigate” and “OnKey” are recommended; but those such as “BatteryNavigate”, “GetUnitInformation”, and “TimerNavigate” should be avoided. Alternatives for these are described below.

The following table summarizes the supported META tag identifier values and their functions. Following the table are individual sections with detailed descriptions of the identifiers. All identifiers are effective only on the page that contains them unless otherwise specified in the description.

<b>META Identifier</b>	<b>Description</b>
Application	Exit the program
Battery	Display on-screen battery information
BatteryNavigate	JavaScript or URL invoked with battery information
Command	Exit the program
CursorPos	Set location of wait cursor
ErrorNavigate	JavaScript or URL invoked on error
GetUnitInformation	JavaScript or URL invoked with device information
HomeKey	Enable/disable home key (F5)
IDA	Invoke IDA Action Code
MoveSIP	Set location of SIP input panel
OnAllKeys	Bind all keys to JavaScript or URL action
OnKey	Bind key to JavaScript or URL action
PowerOn	JavaScript or URL invoked on resume from suspend
Reboot	Invoke a device reset
Scanner	Enable/disable the scanner
ScannerNavigate	JavaScript or URL invoked on successful scan
SetDate	Set the system date
SetTime	Set the system time
Signal	Display on-screen WLAN (RF) signal information
SignalNavigate	JavaScript or URL invoked with WLAN information
SIP	Control the SIP
SIPUp	Show the SIP
TextSize	Set text size factor (zoom)
TimerInterval	Set interval for TimerNavigate
TimerNavigate	JavaScript or URL invoked on TimerInterval
ZebraLabel_Print	Contents of data for printing
ZebraLabel_Complete	JavaScript or URL invoked to report print status

Several special tags contain a JavaScript statement or URL in the content. Depending on the tag, these may contain the text “%s” which is replaced by data which is unique to the tag action. You must have the correct number of replacement placeholders depending on the tag. If the number is incorrect, the tag may not be recognized.

**WARNING:** If the “%s” will be replaced by a text data value, it is important to include single-quotes ( ...'%'s'...) so that the URL is valid and won't contain spaces. Otherwise, the URL may be truncated or not evaluated if the JavaScript is not syntactically correct. In general, it is OK to always include the single-quotes, even for numeric data replacement.

When specifying a URL, it may be any standard form. The JavaScript: form is simply a type of URL. You may also use files local on the device:

```
"file:///Application Data/myapp/errorpage.htm?errno='%s'&msg='%s'"
```

You may also use the proprietary "ida:" type to invoke various actions.

### **3.1 APPLICATION**

The Application tag performs actions which affect the browser application. Currently this only exits the browser.

#### **Syntax**

```
http-equiv="Application"  
content="Quit"
```

#### **Comments**

This tag is supported for compatibility with legacy browsers. It is preferable to use the IDA Action Code "IDA\_PROGRAM\_EXIT" via an "ida:" URL or a PostIDA call.

#### **Example**

```
<html>  
<head>  
<meta http-equiv="Application" content="Quit">  
</head>  
<body>  
This message should not be visible.  
</body>  
</html>
```

### **3.2 BATTERY**

The Battery tag is used to configure the on-screen battery strength meter. This meter overlays the HTML content and is updated at a specified interval. You can change the location and style of the meter. Available styles include a horizontal or vertical single bar meter or filled battery icon. The meter may be repositioned by a "touch and drag" stylus action if dragging is not disabled.

#### **Syntax**

```
http-equiv="Battery"  
content="Show"  
"Hide"  
"Right"  
"Left"  
"Top"  
"Bottom"  
"x=n"  
"y=m"  
"HBattery"  
"VBattery"  
"AllowDrag"  
"NoDrag"
```

Where  $x$  and  $y$  are the screen coordinates of the upper left corner of the meter icon. The screen coordinate (0, 0) is in the upper left corner of the screen with  $x$  increasing to the right and  $y$  increasing downward. Right, Left, Top, and Bottom change the orientation of the meter and the placement of the icon. HBattery enables a horizontal battery icon filled according to strength. VBattery enables a vertical battery icon filled according to strength. AllowDrag will allow the user to drag the meter, whereas NoDrag will prevent dragging.

### Comments

This tag is used to control the battery meter display. The meter may also be controlled within the CETerm configuration, independent of the Battery tag. The Battery tag will always override the internal configuration and will persist until changed by another Battery tag. Within the CETerm configuration you can specify the update interval and a notification when the strength falls below a designated level.

### Example

```
<html>
<head>
<meta http-equiv="Battery" content="Show">
<meta http-equiv="Battery" content="HBattery">
<meta http-equiv="Battery" content="x=200">
<meta http-equiv="Battery" content="y=20">
</head>
<body>
The horizontal battery icon should be visible.
...
</body>
</html>
```

## **3.3 BATTERYNAVIGATE**

The BatteryNavigate tag causes the specified JavaScript or URL to be invoked with battery information on a regular interval. The interval can be specified within a CETerm configuration dialog.

**TIP:** This tag is supported for compatibility with legacy browsers. We recommend using the standard browser method `setTimeout()` together with the Naurtech Device.`GetBatteryInfo()` method rather than the BatteryNavigate META tag. The property “`device.battery.level`” can also be read with `CETerm.GetProperty()` if only the level is monitored. See the CETerm Scripting Guide for more details.

## Syntax

```
http-equiv="BatteryNavigate"
content="javascript:OnBattery('%s', '%s', '%s', '%s');"
<!-- or -->
content="/bat.htm?AC='%s'&strength='%s'&backup='%s'&chemistry='%s'"
```

Where the “%s” are replaced with (1) AC line status, (2) main battery strength as a percentage, (3) backup battery strength as a percentage, and (3) the chemistry of the battery.

## Comments

This tag will work with or without a visible battery meter. The strength normally ranges from 0 to 100. The special strength value of -1 indicates that the strength cannot be determined. If you use a URL for the action, in most cases, the URL will navigate away from the current page rather than repeatedly calling a JavaScript method.

## Example

```
<html>
<head>
<meta http-equiv="BatteryNavigate"
  content="javascript:OnBattery('%s', '%s', '%s', '%s');">
</head>
<body>
<script language=javascript>
function OnBattery( ACstate, strength, backup, chemistry )
{
  if (strength == -1)
  {
    alert("Unable to determine battery strength.");
  }
  else
  {
    alert("Battery strength = " + strength);
  }
}
</script>
...
</body>
</html>
```

### **3.4 COMMAND**

The Command tag performs actions which affect the browser application. Currently this only exits the browser.

#### **Syntax**

```
http-equiv="Command"  
content="exit"
```

#### **Comments**

This tag is supported for compatibility with legacy browsers. It is preferable to use the IDA Action Code "IDA\_PROGRAM\_EXIT" via an "ida:" URL or a PostIDA call.

#### **Example**

```
<html>  
<head>  
<meta http-equiv="Command" content="exit">  
</head>  
<body>  
This message should not be visible.  
</body>  
</html>
```

### **3.5 CURSORPOS**

The CursorPos tag is used to reposition the "busy-cursor" when it is visible. The busy cursor may appear while waiting for pages to load. Use CursorPos to move the busy-cursor from the default location at the center of the screen

#### **Syntax**

```
http-equiv="CursorPos"  
content="x=n"  
       "y=m"
```

Where x and y are the screen coordinates with (0, 0) in the upper left corner of the screen and x increasing to the right and y increasing downward.

#### **Comments**

The new position only applies to the page loading busy-cursor; other busy-cursors will be unaffected. The new position will apply until changed with another CursorPos tag. Not all devices support cursor repositioning.

#### **Example**

```
<html>
<head>
<meta http-equiv="CursorPos" content="x=0">
<meta http-equiv=" CursorPos" content="y=0">
</head>
<body>
Busy cursor in upper left corner.
...
</body>
</html>
```

### **3.6 ERRORNAVIGATE**

The ErrorNavigate tag directs error messages to a JavaScript method or to a URL.

#### **Syntax**

```
http-equiv="ErrorNavigate"
content="javascript:MyErrorHandler('%s', '%s');"
<!-- or -->
content="http://10.1.1.8/errorpage.htm?errno=%s&msg='%s'"
<!-- or to hide errors -->
content="javascript:var HideErrors='%s%s';"
```

The first “%s” is replaced by an error number and the second “%s” by an error message.

#### **Comments**

This tag should be the first special META tag defined on the page, but not before any JavaScript methods that may be invoked by the content. If this tag is not specified, errors are reported via popup messages.

#### **Example**

```
<html>
<head>
<script language=javascript>
function MyErrorHandler(errno, msg)
{
    alert( "Error Number=" + errno + "\nMessage=" + msg );
}
</script>
<meta http-equiv="ErrorNavigate"
    content="javascript:MyErrorHandler(%s, '%s');">
</head>
<body>
...
</body>
</html>
```

### **3.7 GETUNITINFORMATION**

The GetUnitInformation tag reports device and client information to the host or user.

**TIP:** This tag is supported for compatibility with legacy browsers. We recommend using a standard browser onload event handler together with the properties “device.serialnumber”, “device.deviceid”, and “app.version” rather than the GetUnitInformation META tag. The properties can be read with CETerm.GetProperty() method. See the CETerm Scripting Guide for more details.

#### **Syntax**

```
http-equiv="GetUnitInformation"
content="javascript:ReportInfo('%s', '%s', '%s');"
<!-- or -->
content="http://10.1.1.8/info.htm?serial='%s'&uuid='%s'&version='%s'"
```

The first “%s” is replaced by the device serial number, the second %s by the Windows CE device UUID, and third “%s” by the Web Browser version.

#### **Comments**

This tag is supported for compatibility with other browsers. It is preferable to use the CETerm.GetProperty() method of the CETerm Automation Object.

#### **Example**

```
<html>
<head>
<meta http-equiv="GetUnitInformation"
      content="javascript:ReportInfo('%s', '%s', '%s');" />
</head>
<body>
This page reports unit information.
...
<script language=javascript>
function ReportInfo(serial, uuid, version)
{
  alert( "Serial Number=" + serial +
        "\nUUID=" + uuid +
        "\nVersion=" + version );
}
</script>
</body>
</html>
```

### **3.8 HOMEKEY**

The HomeKey tag enables the “home key” (F5) to navigate to the current home URL.

#### **Syntax**

```
http-equiv="HomeKey"  
content="Enabled"  
      "Disabled"
```

#### **Comments**

This tag is supported for compatibility with other browsers. It is preferable to use the IDA Action Code “IDA\_URL\_HOME” via an “ida:” URL or a PostIDA call.

This action can also be achieved with the “OnKey” tag. This tag remains in effect until explicitly changed or the session ends.

#### **Example**

```
<html>  
<head>  
<meta http-equiv="HomeKey" content="Enabled">  
</head>  
<body>  
Press F5 to navigate to the home URL.  
</body>  
</html>
```

### **3.9 IDA**

The IDA tag performs a wide range of actions to control the device and the client.

#### **Syntax**

```
http-equiv="IDA"  
content="IDA_symbolicname"
```

#### **Comments**

This tag offers rich functionality. It is used primarily when an action is needed upon loading a page. Alternatively, an “onload” handler can be used in the BODY element to perform actions via JavaScript.

A list of IDA Action Codes and their description can be found in the CETerm Scripting Guide. All IDA symbolic names must be in upper case.

Any number of IDA meta tags may be specified on a single page. They are acted upon sequentially when parsed.

### Example

```
<html>
<head>
<meta http-equiv="IDA" content="IDA_SIP_SHOW">
</head>
<body>
The Soft Input Panel (SIP) should be visible.
</body>
</html>
```

## **3.10 MoveSIP**

The MoveSIP tag is used to reposition the Soft Input Panel (SIP). MoveSIP is not recommended for hiding the SIP off-screen. Use the SIP lockdown features within CETerm to prevent the SIP from popping up when not wanted.

**WARNING:** MoveSIP may move the SIP to a non-visible location. Usually, entering the CETerm configuration dialogs will temporarily restore the SIP to the default location. Also, window and scroll behavior may be erratic on Windows Mobile devices with the SIP in a non-standard location.

### Syntax

```
http-equiv="MoveSIP"
content="x=n"
      "y=m"
```

Where x and y are the screen coordinates with (0,0) in the upper left corner of the screen and x increasing to the right and y increasing downward.

### Comments

Use MoveSIP to move the default SIP location.

### Example

```
<html>
<head>
<meta http-equiv="MoveSIP" content="x=0">
```

```
<meta http-equiv="MoveSIP" content="y=240">
</head>
<body>
SIP is shifted down a bit.
...
</body>
</html>
```

### **3.11 ONALLKEYS**

The OnAllKeys tag assigns a single JavaScript action or URL to all hardware keys on the handheld device. The action will take place regardless of the focus location on the page.

#### **Syntax**

```
http-equiv="OnAllKeys"

content="javascript:myKeyAction(%s);"
<!-- or -->
content="http://10.1.1.8/inventory.htm?key=%s"
```

The “%s” is replaced by the Windows CE “Virtual Key Code” (VK) value of the key pressed.

#### **Comments**

See Appendix 1 or the CETerm Scripting Guide for a list of Virtual Key Codes, their symbolic names, hexadecimal representation, and the typical keyboard name. If an OnKey tag has been specified for an individual key, that tag’s action will be invoked in place of the OnAllKeys action. See the OnKey tag for additional information.

#### **Example**

```
<html>
<head>
<meta http-equiv="OnAllKeys"
  content="javascript:myKeyAction(%s);" <!-- All keys -->
</head>
<body>
Main Menu<br>
1. Cycle Count<br>
2. Inventory<br>
3. Receiving<br>
Select an action:<br>
Press 'A' to check version.<br>
...
```

```
<script language=javascript>
function myKeyAction(vkcode)
{
    alert( "Key pressed =" + vkcode);
}
</script>
</body>
</html>
```

### **3.12 ONKEY**

The OnKey tag assigns a JavaScript action or URL to hardware keys on the handheld device. The action will take place regardless of the focus location on the page.

#### **Syntax**

```
http-equiv="OnKey0xZZ"
           "OnKeyDDD"
           "OnKeyIgnoreInText"
           "OnKeyVK_name"
           "OnKey_name"

content="javascript:myKeyAction();"
<!-- or -->
content="http://10.1.1.8/inventory.htm"
```

Where ZZ is a two digit hexadecimal number that represents the Windows CE “Virtual Key Code” (VK) for a physical key, or “DDD” is up to 3 decimal digits representing the VK code, or “name” is the portion of the Virtual Key symbolic name after the underscore.

#### **Comments**

See Appendix 1 or the CETerm Scripting Guide for a list of Virtual Key Codes, their symbolic names, hexadecimal representation, and the typical keyboard name. This tag allows several different formats to specify the VK value. Using the symbolic name format yields HTML that is easier to read and maintain. See OnAllKeys to direct all key input to a single action.

Although most VK codes are uniform across devices, some devices can remap the keyboard at a driver level to change the VK codes. Consult your hardware documentation to understand what VK codes are generated by the keys on your device.

Please note that some keys may be tied to operating system actions and they may not be sent to the running applications, thus they cannot be used. Other

times, they are tied to an action and will still be sent to the application, so you may see side-effects of their use.

All key shift states such as CTRL, ALT, and Shift are ignored by the OnKey action. However, it is possible to use shift states via key remapping as described below. We recommended that you do **not** assign OnKey actions to the CTRL, ALT, or SHIFT keys themselves.

The Naurtech Web Browser has very flexible key remapping features. In most cases, any user action which simulates a key, such as a KeyBar button or scanner post-amble will invoke the OnKey action. For example, you may remap Shift+"1" within the Web Browser key remapping to perform the F1 key action. When Shift+"1" is pressed in a browser session, the OnKey action for F1 will be invoked. Tapping on an "F1" KeyBar button will also invoke the OnKey action.

Unlike the Naurtech Web Browser, some other browsers do not act on OnKey assignments when the focus is in a native text INPUT element or in an ActiveX text component. Under Windows Mobile (PIE browser), the "OnKeyIgnoreInText" tag can be used to ignore the OnKey assignments when the focus is in a native text input element. The content is ignored for the "OnKeyIgnoreInText" identifier and the identifier applies only to the current page.

There is no limit to the number of OnKey assignments within a page.

## Example

```
<html>
<head>
<meta http-equiv="OnKey0x1B"
  content="javascript:document.form[0].clearbutton.click();">
<meta http-equiv="OnKey0x31" content="/cyclecount.htm"><!-- 1 -->
<meta http-equiv="OnKey0x32" content="/inventory.htm"><!-- 2 -->
<meta http-equiv="OnKey0x33" content="/receiving.htm"><!-- 3 -->
<meta http-equiv="OnKey0x41" content="ida:IDA_PROGRAM_ABOUT"><!--A-->
</head>
<body>
Main Menu<br>
4. Cycle Count<br>
5. Inventory<br>
6. Receiving<br>
Select an action:<br>
Press 'A' to check version.<br>
...
<form name="form1">
<input type=text name="scan" value="" size=30><br>
<input type=button name="clearbutton" value="Clear Scanned Data"
  onclick="javascript:document.form1.scan.value='';" >
```

```
</form>  
</body>  
</html>
```

### **3.13 POWERON**

The PowerOn tag specifies an action that will occur when the handheld device resumes operation after a power suspend.

#### **Syntax**

```
http-equiv="PowerOn"  
content="javascript:PowerOnAction();"   
<!-- or -->  
content="http://10.1.1.8/login.htm?mode=resume"
```

#### **Comments**

This tag is useful to set the browser to a consistent URL or state after a suspend/resume cycle. For example, a user authentication can be required to maintain security.

#### **Example**

```
<html>  
<head>  
<meta http-equiv="PowerOn"  
content="http://10.1.1.8/login.htm?mode=resume">  
</head>  
<body>  
...  
</body>  
</html>
```

### **3.14 REBOOT**

The Reboot tag will invoke a warm (soft) or cold (hard) reset of the device.

**WARNING: Do not perform the cold reset unless you are prepared to lose all current settings, data, and add-on programs on the device. Techniques are available to automatically restore settings and programs after cold reset. Refer to your device documentation for details.**

#### **Syntax**

```
http-equiv="Reboot"
```

```
content="Warm"  
    "Cold"
```

## Comments

This tag is supported for compatibility with other browsers. It is preferable to use the IDA Action Code “IDA\_WARMBOOT” via an “ida:” URL or a PostIDA() call.

A warm reset will cause all un-saved work-in-progress to be lost. A cold boot will reset all RAM to factory original configurations. A cold boot will typically clear all network and device settings, user data files, and add-on programs that have not been placed in non-volatile memory.

The reset will occur as soon as the tag is parsed.

## Example

```
<html>  
<head>  
<meta http-equiv="Reboot" content="Warm">  
</head>  
<body>  
This message should not be visible.  
</body>  
</html>
```

## 3.15 SCANNER

The Scanner tag is used to enable or disable the barcode scanner.

### Syntax

```
http-equiv="Scanner"  
content="Enabled"  
    "Disabled"
```

## Comments

This tag is used to enable or disable the scanner when a page is first loaded. You can also use the IDA codes IDA\_SCAN\_RESUME and IDA\_SCAN\_SUSPEND to change the state from an “ida:” URL or dynamically via a PostIDA() call.

We do not support the “AutoEnter” or “AutoTab” content values. These values are un-needed because these and more complex scanner post-ambles can be configured within the Naurtech Web Browser. See the User’s Manual for more information about the scanner configuration.

Under Windows Mobile (PIE browser), the Naurtech Web Browser will advance focus within native HTML text input objects when the Tab entered via a key or post-amble, without any extra handlers.

All other scanner configurations are maintained within the Web Browser configuration dialogs. These settings are session dependent and may be different for different Web browser or TE sessions.

### Example

```
<html>
<head>
<meta http-equiv="Scanner" content="Enabled">
</head>
<body>
The scanner will work on this page.
<form name="form1">
<input type="text" name="scan" value="" size=30><br>
...
</form>
</body>
</html>
```

## **3.16 SCANNERNAVIGATE**

The ScannerNavigate tag directs scanner input to the specified JavaScript method or URL. This tag should be used whenever possible to provide the most robust control of scanner input.

### Syntax

```
http-equiv="ScannerNavigate"
content="javascript:OnScan('%s', '%s', '%s', '%s', '%s');"
content="javascript:OnScan('%s', '%s', '%s');"
<!-- or -->
content="/scan.htm?data='%s'&src='%s'&type='%s'&time='%s'&len='%s'"
content="/scan.htm?data='%s'&type='%s'&time='%s'"
```

There are two variants of the ScannerNavigate command for compatibility with other browsers. The first uses 5 parameters and the second uses 3 parameters. For the 5 parameter version, the “%s” are replaced with (1) barcode data, (2) source scanner name, (3) symbology type, (4) timestamp, and (5) barcode length. For the 3 parameter version, the “%s” are replaced with (1) barcode data, (2) symbology type, and (3) timestamp.

The variant type is determined automatically. If there are fewer than 5 substitution parameters, the 5 parameter order is used to fill the specified

parameters. However, if the `IBrowse_` or `IB_` prefix was found (e.g., `IBrowse_ScannerNavigate`), the 3 parameter order is used to fill up to 3 parameters.

## Comments

Only one `ScannerNavigate` tag is permitted on a page. The `ScannerNavigate` tag should be used to ensure that the scanner data is inserted into the correct input element, or is submitted directly via the URL. The barcode data can be examined, validated and/or edited prior to use.

## Example

```
<html>
<head>
<meta http-equiv="ScannerNavigate"
  content="javascript:OnScan('%s', '%s', '%s', '%s', '%s');">
<script>
  ...Above script from Section 2.5 to define and set 'ext'
</script>
</head>
<body>
<form name=form1>
Fill With First Scan<br>
<input type=text name="scan1" value="" size=30><br>
Fill With Second Scan<br>
<input type=text name="scan2" value="" size=30><br>

<script language=javascript>
function OnScan(data, source, type, time, length)
{
  if (document.form1.scan1.value == "")
  {
    document.form1.scan1.value = data;
  }
  else if (document.form1.scan2.value == "")
  {
    document.form1.scan2.value = data;
  }
  else
  {
    ext.CETerm.PostIDA( "IDA_VIBRATE_500", 0 );
    alert("Form full, scan discarded.");
  }
}
</script>
</form>
</body>
</html>
```

### **3.17 SETDATE**

The SetDate tag is used to set the current system date on the handheld.

#### **Syntax**

```
http-equiv="SetDate"  
content="mm/dd/yyyy"  
        "mm, dd, yyyy"  
        "mm-dd-yyyy"
```

Where mm is the month, dd is the day, and yyyy is the year. The date must be expressed in Coordinated Universal Time (UTC). You must use leading zeros for the day and month if less than 10.

#### **Comments**

If the date format is invalid, this tag is ignored.

#### **Example**

```
<html>  
<head>  
<meta http-equiv="SetDate" content="04/15/2010">  
</head>  
<body>  
...  
</body>  
</html>
```

### **3.18 SETTIME**

The SetTime tag is used to set the current system time on the handheld.

#### **Syntax**

```
http-equiv="SetTime"  
content="hh:mm"  
        "hh.mm"
```

Where hh is the hour, and mm is the minute. The time must be expressed in Coordinated Universal Time (UTC). You must use leading zeros if less than 10.

#### **Comments**

If the time format is invalid, this tag is ignored.

#### **Example**

```
<html>  
<head>
```

```
<meta http-equiv="SetTime" content="12:01">
</head>
<body>
...
</body>
</html>
```

### **3.19 SIGNAL**

The Signal tag is used to configure the on-screen WLAN (RF) signal strength meter. This meter overlays the HTML content and is updated at a specified interval. You can change the location and style of the meter. Available styles include a horizontal or vertical single bar meter and a stepped bar meter. The meter may be repositioned by a “touch and drag” stylus action if dragging is not disabled.

#### **Syntax**

```
http-equiv="Signal"
content="Show"
      "Hide"
      "Right"
      "Left"
      "Top"
      "Bottom"
      "x=n"
      "y=m"
      "Steps"
      "AllowDrag"
      "NoDrag"
```

Where x and y are the screen coordinates of the upper left corner of the meter icon. The screen coordinate (0,0) is in the upper left corner of the screen with x increasing to the right and y increasing downward. Right, Left, Top, and Bottom change the orientation of the meter and the placement of the WLAN icon. Steps enables the step style meter. AllowDrag will allow the user to drag the meter, whereas NoDrag will prevent dragging.

#### **Comments**

This tag is used to control the WLAN (RF) meter display. The meter may also be controlled within the CETerm configuration, independent of the Signal tag. The Signal tag will always override the internal configuration and will persist until changed by another Signal tag. Within the CETerm configuration you can specify the update interval and a notification when the strength falls below a designated level.

## Example

```
<html>
<head>
<meta http-equiv="Signal" content="Show">
<meta http-equiv="Signal" content="Steps">
</head>
<body>
The RF meter should be visible.
...
</body>
</html>
```

## **3.20 SIGNALNAVIGATE**

The SignalNavigate tag causes the specified JavaScript or URL to be invoked with WLAN signal information on a regular interval. The interval can be specified within a CETerm configuration dialog.

**TIP:** This tag is supported for compatibility with legacy browsers. We recommend using the standard browser method `setTimeout()` together with the property `device.rf.strength` rather than the SignalNavigate META tag. The properties `device.rf.strength`, `device.rf.ssid`, and `device.macaddress` can be read with `CETerm.GetProperty()` method. See the CETerm Scripting Guide for more details.

## Syntax

```
http-equiv="SignalNavigate"
content="javascript:OnSignal('%s', '%s', '%s');"
<!-- or -->
content="/signal.htm?strength='%s'&ESSID='%s'&MAC='%s'"
```

Where the "%s" are replaced with (1) signal strength, (2) ESSID, and (3) the MAC address of the device.

## Comments

This tag will work with or without a visible WLAN signal meter. The strength normally ranges from 0 to 100. The special strength value of -2 indicates that the device is not associated with any access point. The special strength value of -1 indicates that the strength cannot be determined. If you use a URL for the action, in most cases, the URL will navigate away from the current page rather than repeatedly calling a JavaScript method.

## Example

```
<html>
<head>
<meta http-equiv="SignalNavigate"
  content="javascript:OnSignal('%s', '%s', '%s');">
</head>
<body>
<script language=javascript>
function OnSignal(strength, ESSID, MACAddress)
{
  if (strength == -2)
  {
    alert("RF radio is not associated.");
  }
  else if (strength == -1)
  {
    alert("Unable to determine RF signal strength.");
  }
  else
  {
    alert("RF Signal strength = " + strength);
  }
}
</script>
...
</body>
</html>
```

## **3.21 SIP**

The SIP tag controls the visibility of the Soft Input Panel (SIP).

### **Syntax**

```
http-equiv="SIP"
content="Show"
       "Hide"
       "Locked"
```

### **Comments**

This tag is used primarily when the SIP should be displayed upon loading a page. Alternatively, you can use an IDA Action Code via an “ida:” URL or a PostIDA call. These latter techniques can be used to change the SIP visibility during the user interaction with the page. The “Locked” value will prevent the SIP from popping up when focus is set to a text input element under Windows Mobile (PIE browser). The locked state can also be set in the CETerm configuration dialogs.

**Example**

```
<html>
<head>
<meta http-equiv="SIP" content="Show">
</head>
<body>
The Soft Input Panel (SIP) should be visible.
</body>
</html>
```

**3.22 SIPUP**

The SIPUP tag makes the Soft Input Panel (SIP) visible.

**Syntax**

```
http-equiv="SIPUp"
content=""
```

**Comments**

This tag is supported for compatibility with other browsers. It is preferable to use the SIP META tag or the IDA Action Code "IDA\_SIP\_SHOW" via an "ida:" URL or a PostIDA call. The content is ignored, but it must be present.

**Example**

```
<html>
<head>
<meta http-equiv="SIPUP" content="">
</head>
<body>
The Soft Input Panel (SIP) should be visible.
</body>
</html>
```

**3.23 TEXTSIZE**

The TextSize tag is used to set an overall text zoom level. The zoom level can also be changed manually with font sizing buttons on the Toolbar or KeyBar.

**Syntax**

```
http-equiv="TextSize"
content="Smallest"
       "Smaller"
       "Medium"
```

```
"Larger"  
"Largest"
```

## Comments

The TextSize tag is somewhat misnamed. Relative text sizes are determined by the HTML content. TextSize applies an overall zoom factor to the sizes set in the HTML. The initial zoom value will be the last value set for the session whether by a TextSize tag or by manual actions.

## Example

```
<html>  
<head>  
<meta http-equiv="TextSize" content="Largest">  
</head>  
<body>  
This is the Largest TextSize zoom.  
...  
</body>  
</html>
```

## 3.24 TIMERINTERVAL

The TimerInterval tag is used to specify the interval between activations of the TimerNavigate action.

## Syntax

```
http-equiv="TimerInterval"  
content="milliseconds"
```

Where milliseconds is the number of milliseconds between activations.

## Comments

The effect of TimerInterval and TimerNavigate tags can be accomplished using the JavaScript `setInterval()` and `setTimeout()` methods on the window DOM object. Using the JavaScript methods is recommended.

## Example

See example under TimerNavigate.

## 3.25 TIMERNAVIGATE

The TimerNavigate tag causes the specified JavaScript or URL to be invoked on a regular interval. The interval is specified with the TimerInterval tag.

**TIP:** This tag is supported for compatibility with legacy browsers. We recommend using the standard browser method `setTimeout()` or `setInterval()` for periodic actions rather than `TimerNavigate` and `TimerInterval`.

## Syntax

```
http-equiv="TimerNavigate"
content="javascript:OnTimer('%s');"
<!-- or -->
content="/timer.htm?time='%s'"
```

Where the “%s” is replaced with the current time in the form hh:mm:ss.

## Comments

The effect of `TimerInterval` and `TimerNavigate` tags can be accomplished using the JavaScript `setInterval()` and `setTimeout()` methods on the `window` DOM object. Using the standard JavaScript methods is recommended.

## Example

```
<html>
<head>
<meta http-equiv="TimerInterval" content="1000">
<meta http-equiv="TimerNavigate"
  content="javascript:OnTimer('%s');">
</head>
<body>
Current time: <div id="timerDiv">junk</div>
<script language=javascript>
function onTimer(time)
{
  timerDiv.innerHTML = time;
}
</script>
...
</body>
</html>
```

### **3.26 ZEBRALABEL\_COMPLETE OR PLSERIESLABEL\_COMPLETE**

The `ZebraLabel_Complete` tag is used to report the status of a print from the `ZebraLabel_Print` tag. The alternative identifier `PLSeriesLabel_Complete` will also be recognized.

**TIP:** This tag is supported for compatibility with legacy browsers. It is preferable to use other techniques for sending print content to a printer. See the advanced topic “Printing from HTML”.

### Syntax

```
http-equiv="ZebraLabel_Complete"  
content="javascript:PrintStatus('%ld');"  
<!-- or -->  
content="http://10.1.1.8/print.htm?status=%ld"
```

### Comments

The status code of the print action will be substituted into the “%ld” location. A status of 0 indicates success, 1 indicates failure. See ZebraLabel\_Print for more details.

### Example

```
<html>  
<head>  
<meta http-equiv="ZebraLabel_Complete"  
  content= "http://10.1.1.8/print.htm?status=%ld"  
<meta http-equiv="ZebraLabel_Print"  
  content="! 0 200 200 581 1\r\nLABEL ... PRINT\r\n">  
</head>  
<body>  
The print should be produced.  
This message should not be visible.  
</body>  
</html>
```

## **3.27 ZEBRALABEL\_PRINT OR PLSERIESLABEL\_PRINT**

The ZebraLabel\_Print tag contains data that is sent to a Zebra printer. The ZebraLabel\_Complete tag must also be present to report the status of the print. The alternative identifier PLSeriesLabel\_Print will also be recognized.

**TIP:** This tag is supported for compatibility with other browsers. It is preferable to use other techniques for sending print content to a printer. See the advanced topic “Printing from HTML”.

### Syntax

```
http-equiv="ZebraLabel_Print"
```

```
content="! 0 200 200 581 1\r\nLABEL ... PRINT\r\n">
```

## Comments

This tag is supported for compatibility with other browsers. It is preferable to use other techniques for sending print content to a printer. See the advanced topic “Printing from HTML”.

The contents must be less than 1024 characters. Any “\r” sequences are replaced by the CR (0x0d) character. Any “\n” sequences are replaced by the NL (0x0a) character. The sequence “\xXX” is replaced by a byte with the hexadecimal value XX. Use “\” for a literal backslash.

The content is sent to the printer that is currently configured under the CETerm Printer tab. This may be a serial attached printer, Bluetooth, or network accessible printer.

## Example

```
<html>
<head>
<!-- NOTE - The content must be on one line -->
<!-- NOTE - It is split here for readability -->
<meta http-equiv=" ZebraLabel_Print "
  content="! 100 200 200 1225 1\r\n
          TEXT 0 2 1 0 Vendor\r\nTEXT 5 2 1 20 Adidas@\r\n
          TEXT 0 2 1 80 Model\r\nTEXT 5 2 1 100 Adidas@\r\n
          TEXT 5 2 1 135 Storm\r\nTEXT 5 0 1 200 Features\r\n
          TEXT 5 0 1 225 Cross-Training\r\n
          TEXT 5 0 1 250 Arch Support\r\n
          TEXT 5 0 1 275 Leather Upper\r\n
          TEXT 5 0 1 300 Cushioned Insole\r\n
          TEXT 4 1 100 435 $61.99\r\n
          TEXT 0 2 1 550 _____\r\n
          TEXT180 4 1 250 700 $61.99\r\n
          TEXT180 5 0 270 740 000000292818\r\n
          BARCODE UPCA 1 1 50 100 750 000000292818\r\n
          TEXT180 5 0 320 910 WHITE/RED\r\n
          TEXT180 5 0 320 935 Available in:\r\n
          TEXT180 0 2 320 1025 _____\r\n
          JOURNAL\r\nPRINT\r\n">
<meta http-equiv="ZebraLabel_Complete"
  content= "http://10.1.1.8/print.htm?status=%ld"
</head>
<body>
The print should be produced.
This message should not be visible.
</body>
</html>
```

## 4.0 Advanced Topics

In this chapter, we discuss several common, but advanced, topics for creating robust Web based applications. We discuss how to overcome some limitations of the native browser behaviors.

### **4.1 NAVIGATING TO PRE-CONFIGURED URLS**

The Naurtech Web Browser allows you to configure pre-defined URL's which can be associated with a hardware key, a KeyBar button, or activated from the within the browser via a link or via JavaScript.

The URL is defined within a User Text string and this text can be submitted for navigation by any of the ways listed above. For example, with the following content in User Text 1:

```
\IDA_URL\file:///Program Files/myhelppage.htm\r
```

You can navigate to this local help page by activating the IDA code IDA\_USTRING\_1 which has the friendly name "Text 1". You can associate a hardware key or a KeyBar button to this IDA code. Alternatively, you could use any of the ways listed in the IDA Action Codes section to trigger the navigation. For example the following link will send you to the help page:

```
<a href="ida:IDA_USTRING_1">Show Help Page</a>
```

Be sure to provide a way to navigate back to the application from your help page. Note the required "IDA\_URL" at the beginning of the URL and the "\r" at the end. If your URL contains a literal backslash, it must be escaped with a second backslash "\\".

It is possible to change the contents of User Text from within JavaScript with the CETerm.SetProperty() method of the CETerm Automation Object. Thus, this storage can be used to maintain some persistent and device-local information.

### **4.2 CONTROLLING THE SCANNER**

We have shown how the Scanner and ScannerNavigate META tag identifiers can be used to control the scanner. You can also use IDA codes to provide additional control.

The scanner can be enabled or disabled via an IDA code. You can also activate a "soft trigger" on most scanner equipped handhelds. This will start the scanner without the user needing to hold the trigger. This mode is useful if the application is driving the scanner in a "read loop" until a desired number of scans are completed.

```
<meta http-equiv="Scanner" content="Enabled">
<meta http-equiv="ScannerNavigate"
  content="Javascript:OnScan('%s','%s','%s','%s','%s');">
...
<script language=javascript>
// Put this <script> element in the <head> of the page.
// Resolve ext reference one time, when page loads.
// WARNING: Make sure ext is not already used by your scripts.
// Examples which reference top-level objects:
// ext.CETerm.PostIDA( "IDA_SIP_SHOW", 0 ); // show SIP
// ext.OS.File.Append( "\\myfile.txt", "content" );
// var spl = ext.Device.SerialPort(1); // Get a SerialPort object
//
var ext = null; // global variable, not declared in a function
if (typeof external === "object")
{
  // Windows CE
  // external is already defined in global namespace
  ext = external;
}
else
{
  // Windows Mobile
  // Create CEBrowseX for top-level object access
  ext = new ActiveXObject( "CebrowseX.IdaCtl" );
}
</script>
...
<input type=button name="autoscan" value="Auto Scan"
onclick="autoscan();">
...
<script language=javascript>
function autoscan()
{
  if (autoscantrigger)
  {
    autoscantrigger = 0;
  }
  else
  {
    autoscantrigger = 1;
    ext.CETerm.PostIDA( "IDA_SCAN_TRIGGER", 0 );
  }
}

function OnScan(data, source, type, time, length)
{
  if (document.form1.scan1.value == "")
  {
    document.form1.scan1.value = data;
  }
  else if (document.form1.scan2.value == "")
```

```
    {
        document.form1.scan2.value = data;
    }
    else if (document.form1.scan3.value == "")
    {
        document.form1.scan3.value = data;
        autoscantrigger = 0;
    }
    else
    {
        autoscantrigger = 0;
        ext.CETerm.PostIDA( "IDA_VIBRATE_500", 0 );
        alert("Form full, scan discarded.");
    }

    if (autoscantrigger)
    {
        ext.CETerm.PostIDA( "IDA_SCAN_TRIGGER", 0 );
    }
}
</script>
```

### **4.3 INPUT FOCUS AND THE TAB KEY**

One of the first limitations you are likely to notice about Pocket Internet Explorer in Windows Mobile is that the Tab key does not always move the input focus as you expect from using the desktop Internet Explorer. Pocket IE on Windows Mobile platforms has the most limited behavior (see Section 1.2).

The Naurtech Web Browser provides enhanced focus control on Windows Mobile platforms. If you only need to move between native HTML text input elements, then the Tab key should perform as you expect from the desktop. The Tab key will also work as expected on Windows CE platforms.

Some other vendors provide a text input control to manage the focus under Windows Mobile. We don't recommend using ActiveX controls for input because of the additional development and maintenance costs.

**NOTE:** The Naurtech TextX input component is no longer provided or supported. The component is no longer needed because the Naurtech Web Browser can advance the focus using the TAB key and tracking can be performed with standard onfocus event handlers.

The following examples show how to track the focus and advance the focus when the Tab key is pressed.

### 4.3.1 Windows CE (IE6CE) Example

This first example uses native HTML text input on Windows CE. Note the use of “document.activeElement” in this example:

```
<html>
<head>
<title>Naurtech Windows CE Tab Demo Page</title>
<meta http-equiv="Scanner" content="Enabled">
<meta http-equiv="ScannerNavigate"
content="Javascript:OnScan('%s','%s','%s','%s','%s');">
<meta http-equiv="OnKey0x0D"
content="Javascript:document.form1.clear.click();">
</head>
<body scroll=no onload="Javascript:document.form1.scan1.focus();">
<form name=form1>
<center>
<font size=+2>
Naurtech Tab Demo<br>
Windows CE<br>
</font>
<br>
Focus starts in first input<br>
<input type=text name="scan1" value="" size=30
  onkeypress="myonkey();"><br>
<input type=text name="scan2" value="" size=30
  onkeypress="myonkey();"><br>
<input type=text name="scan3" value="" size=30
  onkeypress="myonkey();"><br>
<input type=button name="clear" value="Clear Data"
  onclick="myclear();" onkeypress="myonkey();">
</center>
</form>

<script for="document" event=onkeypress>
// IMPORTANT: This handler is used when focus is
// not already in an input object
myonkey();
</script>

<script language=javascript>

// Handle the key event
function myonkey()
{
  if (window.event.keyCode == 9) // look for tab key
  {
    nextfield( document.activeElement );

    window.event.cancelBubble = true;
  }
}
```

```
// Move from the current field to the next field
function nextfield( current )
{
    if (current == document.form1.scan1)
    {
        document.form1.scan2.focus();
    }
    else if (current == document.form1.scan2)
    {
        document.form1.scan3.focus();
    }
    else
    {
        document.form1.scan1.focus();
    }
}

// Clear the fields
function myclear()
{
    document.form1.scan1.value = "";
    document.form1.scan2.value = "";
    document.form1.scan3.value = "";
    ring();
    document.form1.scan1.focus();
}

// Play a sound
function ring()
{
    external.OS.PlayTone(10, 1000, 200);
}

// Handle the scan data
function OnScan(data, source, type, time, length)
{
    var current = document.activeElement;
    if (current == document.form1.scan1 ||
        current == document.form1.scan2 ||
        current == document.form1.scan3)
    {
        current.value = data;

        // NOTE: Any postamble configured in CETerm will not be
        // NOTE: seen when using the ScannerNavigate feature.
        // NOTE: The following method advances the focus.
        nextfield( current );
    }
    else
    {
        alert("Scan discarded, focus is not in a scan field.");
    }
}
}
```

```
</script>
</body>
</html>
```

### 4.3.2 Windows Mobile (PIE) Example

This example tracks the last input focus using a hidden form variable because the “document.activeElement” attribute is not available under PIE:

```
<html>
<head>
<title>Naurtech Windows Mobile Tab Demo Page</title>
<meta http-equiv="Scanner" content="Enabled">
<meta http-equiv="ScannerNavigate"
      content="Javascript:OnScan('%s','%s','%s','%s','%s');">
<meta http-equiv="OnKey0x0D"
      content="Javascript:document.form1.clear.click();">

<script language=javascript>
// Actions when focus enters a field
function enterFocusField( name )
{
  // Save new field name
  document.form1.focusElement.value = name;
}
</script>

</head>
<body scroll=no onload="document.form1.scan1.focus();">
<form name=form1>
<!-- Hidden field to track focus -->
<input type="hidden" id="focusElement" name="focusElement"
      value="" />
<center>
<font size=+2>
Naurtech Tab Demo<br>
Windows Mobile<br>
</font>
<br>
Focus starts in first input<br>
<input type=text name="scan1" value="" size=30
      onfocus="enterFocusField('scan1');"><br>
<input type=text name="scan2" value="" size=30
      onfocus="enterFocusField('scan2');"><br>
NoScan<input type=text name="noscan" value="" size=30
      onfocus="enterFocusField('noscan');"><br>
<input type=text name="scan3" value="" size=30
      onfocus="enterFocusField('scan3');"><br>
<input type=button name="clear" value="Clear Scan Data"
      onclick="myClear();">
```

```
</center>
</form>

<script language=javascript>
// Move from the current scan field to the next scan field
function nextField( current )
{
    if (current === "scan1")
    {
        document.form1.scan2.focus();
    }
    else if (current === "scan2")
    {
        document.form1.scan3.focus();
    }
    else if (current === "scan3")
    {
        document.form1.scan1.focus();
    }
}

// Clear the fields
function myClear()
{
    document.form1.scan1.value = "";
    document.form1.scan2.value = "";
    document.form1.scan3.value = "";
    ring();
    document.form1.scan1.focus();
}

// Play a sound
function ring()
{
    var ext = new ActiveXObject( "Cebrowsex.IdaCtl" );
    ext.OS.PlayTone(10, 1000, 200);
}

// Handle the scan data
function OnScan(data, source, type, time, length)
{
    var current = document.form1.focusElement.value;

    if (current === "scan1" ||
        current === "scan2" ||
        current === "scan3")
    {
        document.form1[current].value = data;

        // NOTE: Any postamble configured in CETerm will not be
        // NOTE: seen when using the ScannerNavigate feature.
        // NOTE: The following method advances the focus.
        nextField( current );
    }
}

```

```
    }
    else
    {
        alert("Scan discarded, focus is not in a scan field.");
    }
}
</script>
</body>
</html>
```

## **4.4 SESSION LAUNCHER**

One interesting use of the Web Browser is as a launcher for other browser or terminal emulation sessions. The following example shows how a static Web page on the device could be used to activate other sessions. This example assumes that this Web page is displayed under Session 4. Each of the other sessions must be configured for the desired activity:

```
<html>
<head>
<title>Naurtech Launch Page</title>
<meta http-equiv="PowerOn" content="Javascript:poweron();">
<meta http-equiv="Scanner" content="Disabled">
<meta http-equiv="OnKey0x70"
    content="Javascript:startsession(1);"><!-- F1 -->
<meta http-equiv="OnKey0x71"
    content="Javascript:startsession(2);"><!-- F2 -->
<meta http-equiv="OnKey0x72"
    content="Javascript:startsession(3);"><!-- F3 -->
<script language=javascript>
// Put this <script> element in the <head> of the page.
// Resolve ext reference one time, when page loads.
// WARNING: Make sure ext is not already used by your scripts.
// Examples which reference top-level objects:
// ext.CETerm.PostIDA( "IDA_SIP_SHOW", 0 ); // show SIP
// ext.OS.File.Append( "\\myfile.txt", "content" );
// var sp1 = ext.Device.SerialPort(1); // Get a SerialPort object
//
var ext = null; // global variable, not declared in a function
if (typeof external === "object")
{
    // Windows CE
    // external is already defined in global namespace
    ext = external;
}
else
{
    // Windows Mobile
    // Create CEBrowseX for top-level object access
    ext = new ActiveXObject( "CebrowseX.IdaCtl" );
}
}
```

```
</script>
</head>
<body>
<form name=form1>
<font size=9>
<center>
Main Menu<br>
</center>
<a href="javascript:startsession(1);">F1. Pick</a><br>
<a href="javascript:startsession(2);">F2. Cycle Count</a><br>
<a href="javascript:startsession(3);">F3. Receive</a><br>
<br>
</font>
</form>

<script language=javascript>
function poweron()
{
    // Navigate to Main Menu on resume
    sendida( "IDA_SESSION_S4", 0 );
}

function startsession(id)
{
    if (id >= 1 && id <=4)
    {
        // Switch to session
        sendida( "IDA_SESSION_S" + id, 0 );
        // Connect if not connected
        sendida( "IDA_SESSION_CONNECT", id );
        return;
    }
    else
    {
        alert("Unsupported session.");
    }
}

function sendida(ida, s)
{
    ext.CETerm.SendIDA( ida, s );
}
</script>
</body>
</html>
```

## **4.5 HOW TO IDENTIFY THE CURRENT BROWSER**

The Naurtech Web Browser returns the same User-Agent HTTP value as does the standard Windows browser on the device. This is necessary to indicate the fundamental capabilities of the browser to the Web server. There are other ways for the Web application to determine if the Naurtech Web Browser is the client.

We recommend using the `GetProperty()` function of the `CETerm Automation Object` to detect `CETerm` and identify the version if needed. Other browsers will not be able to provide a valid `CETerm` object reference.

```
<script language=javascript>
// Put this <script> element in the <head> of the page.
// Resolve ext reference one time, when page loads.
// WARNING: Make sure ext is not already used by your scripts.
// Examples which reference top-level objects:
// ext.CETerm.PostIDA( "IDA_SIP_SHOW", 0 ); // show SIP
// ext.OS.File.Append( "\\myfile.txt", "content" );
// var sp1 = ext.Device.SerialPort(1); // Get a SerialPort object
//
var ext = null; // global variable, not declared in a function
if (typeof external === "object")
{
    // Windows CE
    // external is already defined in global namespace
    ext = external;
}
else
{
    // Windows Mobile
    // Create CEBrowseX for top-level object access
    ext = new ActiveXObject( "CebrowseX.IdaCtl" );
}
</script>
...
<script language=javascript>
function checkversion()
{
    if (ext != null && ext.CETerm != null)
    {
        version = ext.CETerm.GetProperty( "app.version" );
        // check the version value, return info to host, etc.
        ...
    }
}
}
```

You could also use the `GetUnitInformation META` tag identifier to retrieve the version number and send this to the host. As of this writing, the current version number is “5.7.0”. Most other browsers will not support the `GetUnitInformation` feature and your HTML must be prepared to return empty values. We recommend avoiding `GetUnitInformation` unless it is needed by a legacy web application.

```
<html>
<head>
<meta http-equiv="GetUnitInformation"
      content="Javascript:saveunitinfo('%s','%s','%s');">
<script language=javascript>
var serial;
var ceuuid;
var version;

function saveunitinfo(serialarg, ceuuidarg, versionarg)
{
  serial = serialarg;
  ceuuid = ceuuidarg;
  version = versionarg;
}
</script>
</head>
<body onload="javascript:checkversion();">
....check version variable and return info to host....
</body>
```

## **4.6 DEVICE INFORMATION**

You can retrieve additional information about the device configuration using the `CETerm.GetProperty()` method. See the *CETerm Scripting Guide* for details.

## **4.7 SYMBOL WEB CLIENT**

An early browser developed for Symbol devices running the Palm OS had some extensions for controlling the scanner and printing. These extensions were implemented through custom attributes on standard HTML tags and custom HTML tags. The Naurtech Web Browser running on Windows CE platforms supports these extensions. These extensions are not supported on Windows Mobile devices. These extensions are supported for compatibility with existing Web based applications. New Web applications should use the META tags for scanner control.

On the text INPUT element we support the “stiscan” and “stisubmit” attributes. If present, the stiscan attribute controls which symbologies may be inserted into the input element. If no value is specified, all symbologies are allowed. If an empty string is specified, no symbologies are allowed. When the “stisubmit” attribute is present, the enclosing form will be submitted after scanned data is inserted in the element.

```
<input type=text name=upc size=12 maxlength=16  
stiscan="ABCDEFGHijklmn" stisubmit>
```

The custom element BEEP is supported to sound a tone:

```
<NAURTECH:beep frequency="f" repeat="n" duration="d">
```

where f is in Hz, n is a count, and d is in milliseconds. The handheld device must support a tone generator. The “NAURTECH:” namespace label is required.

The custom element PRINT is supported to send print content to the printer.

```
<NAURTECH:print name="printer" type="N" announce="yes" retries=3>  
Print content here\n\r  
</print>
```

where “printer” is the destination name of the printer, and type is the connection type. The “NAURTECH:” namespace label is required. The following connection types are supported:

Connection Type	Name Values	Description
Q	Name is the Windows print queue name	Print to Windows print queue
N	Printer IP address or hostname and port. For example, “192.168.1.101:6101”	Print directly to port
dddd	Same as ‘N’ but port specified as dddd.	Print directly to port
I, B, S, (other)	(ignored)	Print to configured printer for current session.

Standard escape characters such as “\r”, “\n”, and “\xXX” will be substituted in the print contents. Literal line breaks in the content are ignored, you must use “\r\n” if the printer language requires a line break.

You can set the values of the currently configured network printer using the SetProperty() method on the CETerm Automation Object.

## 5.0 Printing from HTML

There are numerous ways to print from the Naurtech Web Browser. After a printer is configured, print content may be specified via:

- a special META tag,
- the CEBrowseX PrintString() method,
- the “external” Print() method,
- the custom PRINT tag,
- or direct serial port printing.

Printing can also be done with any ActiveX control designed to print from a browser.

The Naurtech Web Browser maintains the printer configuration within the Session configuration dialogs. Here you may specify a serial attached, Bluetooth, IrDA, or network attached printer. We support both Windows Print Queues and direct-to-port printing for network printers. See the CETerm User’s Manual for more details on configuring a printer.

All of the techniques for specifying print content allow common escape sequences to be embedded which will be converted to non-printable characters. These include the carriage return (CR – “\r”), linefeed (LF – “\n”), and general hexadecimal bytes (“\xXX”).

### **5.1 PRINTING WITH A META TAG**

See the “ZebraLabel\_Print” and “ZebraLabel\_Complete” identifiers in the META tag section for details on initiating a print from a META tag.

### **5.2 PRINTSTRING AND PRINT METHODS**

The PrintString() method on the CEBrowseX control and the Print() method on the “external” object behave similarly. In both cases, a string is constructed and sent to the current printer.

```
<a href="Javascript:myprint();">Test External Print</a>
<script language=javascript>
function myprint()
{
    external.Print(
        "! 100 200 200 1225 1\r\n" +
        "TEXT 0 2 1 0 Vendor\r\n" +
        ...
        "JOURNAL\r\n" +
        "PRINT\r\n"
    );
}
</script>
```

or, using PrintString() on CEBrowseX

```
<a href="Javascript:myprint();">Test External Print</a>
<script language=javascript>
function myprint()
{
    var ext = new ActiveXObject( "CebrowseX.IdaCtl" );
    ext.PrintString(
        "! 100 200 200 1225 1\r\n" +
        "TEXT 0 2 1 0 Vendor\r\n" +
        ...
        "JOURNAL\r\n" +
        "PRINT\r\n"
    );
}
</script>
```

### **5.3 NAURTECH:PRINT TAG**

The custom PRINT tag is supported for compatibility with other browsers. See the Symbol Web Client section for a discussion of this tag.

### **5.4 DIRECT SERIALPORT PRINTING**

Full bi-directional control of any real or virtual (e.g., Bluetooth) serial port is possible from CETerm Scripting. Access is provided by the Device.SerialPort() objects. You can construct print output within a web page and direct it to any printer accessible via a serial port. See the CETerm Scripting Guide for details about the SerialPort object.

### **5.5 ACTIVEX PRINTING CONTROLS**

There are third-party ActiveX controls available to send print content to a printer. These are available from handheld manufacturers, printer manufacturers, and other third-party sources. All controls which follow accepted standards will work within the Naurtech Browser. Currently we do not recommend any specific control.

## 6.0 CEBrowseX Control

The Naurtech CEBrowseX control is used to obtain references to the CETerm Automation Objects within the Windows Mobile PIE browser. This control is installed when you install the Naurtech Web Browser.

Although CEBrowseX is also installed and can be used on Windows CE platforms, it is not required because of the pre-defined Document Object Model (DOM) "external" reference which provides references to the Automation Objects.

**NOTE:** Previously, the CEBrowseX control contained some of the same functionality as the CETerm object. The functionality was deprecated in CETerm V5.1 and is no longer supported. Existing Web pages may continue to work but should be converted to use only the references to the automation objects.

**TIP:** See the CETerm Scripting Guide for full details on the CETerm Automation Objects, their methods and properties. Scripting in CETerm offers additional capabilities to enhance your Web based applications.

### SYNTAX

```
<script language=javascript>
  // Create CEBrowseX for top-level object access
  var ext = new ActiveXObject( "CebrowseX.IdaCtl" );

  // Use ext reference to CEBrowseX
  ext.CETerm.PostIDA( "IDA_SESSION_S1", 0 );
</script>
```

The CEBrowseX control can also be instantiated with an <object> tag but the JavaScript technique is preferred.

```
<object id="CEBrowseX"
  classid="clsid:D14943BD-4900-453E-8582-725F21A57E0C"
  height=0, width=0>
  ...
  <script language=javascript>
  function mybeep()
  {
    document.all.CEBrowseX.CETerm.PostIDA( "IDA_BEEP_LOUD", 0 );
  }
  </script>
```

The CEBrowseX control has no visible display, but you must specify height and width of zero to prevent the object from consuming space on the page.

## **CLASSID**

The CEBrowseX CLASSID is:

```
CLASSID="clsid:D14943BD-4900-453E-8582-725F21A57E0C"
```

## **METHODS**

The following methods are available. The CETerm Automation Objects are available as CEBrowseX properties.

<b>Method</b>	<b>Action</b>
PrintString	Send content to a printer (CEBrowseX only)
Print	Send content to a printer (external only)

*Status = PrintString( printData )*

PrintString will send the printData to the currently configured printer. PrintString is only available on the CEBrowseX object.

The printData string may contain escape characters for CR (\r), Newline (\n), and hexadecimal bytes (\xXX).

**NOTE:** The Print method is not actually part of CEBrowseX but is documented here because it is equivalent to PrintString. If using CEBrowseX on a Windows Mobile platform (PIE), use PrintString. If using “external” on a Windows CE platform (IE6CE), use Print.

*Status = Print( printData )*

Print will send the printData to the currently configured printer. Print is only available on the “external” DOM reference under Windows CE (IE6CE browser).

The printData string may contain escape characters for CR (\r), Newline (\n), and hexadecimal bytes (\xXX).

## **PROPERTIES**

The CEBrowseX properties are used to return the CETerm Automation Objects.

<b>Property</b>	<b>Description</b>
CETerm	CETerm automation object (read only)
Device	Device automation object (read only)
OS	OS automation object (read only)

## **EVENTHANDLERS**

The CEBrowseX control has no event handlers.

## Appendix 1 - Virtual Key Codes

This appendix contains a list of Windows CE Virtual Key Codes (VK) which are used with the OnKey META tag.

Notice that there is no case distinction of the alphabetic keys. Also, note that the symbols on the tops of the digit keys are not listed because they are a shift state of the digit keys.

**WARNING:** The keys on some devices do not send standard VK values. Often, the printed keycap label does not reflect the value sent to applications. If your key mappings are not working as expected, you may need to refer to device documentation or the "Trap" feature in the CETerm "New Key" remapping configuration dialog to identify the correct value.

Symbolic Name	Hexadecimal Value	Keyboard Equivalent
VK_BACK	0x08	BACKSPACE key
VK_TAB	0x09	TAB key
VK_CLEAR	0x0C	CLEAR key
VK_RETURN	0x0D	ENTER key
VK_SHIFT	0x10	SHIFT key
VK_CONTROL	0x11	CTRL key
VK_MENU	0x12	ALT key
VK_PAUSE	0x13	PAUSE key
VK_CAPITAL	0x14	CAPS LOCK key
VK_ESCAPE	0x1B	ESC key
VK_SPACE	0x20	SPACEBAR
VK_PRIOR	0x21	PAGE UP key
VK_NEXT	0x22	PAGE DOWN key
VK_END	0x23	END key
VK_HOME	0x24	HOME key
VK_LEFT	0x25	LEFT ARROW key
VK_UP	0x26	UP ARROW key
VK_RIGHT	0x27	RIGHT ARROW key
VK_DOWN	0x28	DOWN ARROW key
VK_SELECT	0x29	SELECT key
VK_EXECUTE	0x2B	EXECUTE key
VK_SNAPSHOT	0x2C	PRINT SCREEN key

<b>Symbolic Name</b>	<b>Hexadecimal Value</b>	<b>Keyboard Equivalent</b>
VK_INSERT	0x2D	INS key
VK_DELETE	0x2E	DEL key
VK_HELP	0x2F	HELP key
VK_0	0x30	0 key
VK_1	0x31	1 key
VK_2	0x32	2 key
VK_3	0x33	3 key
VK_4	0x34	4 key
VK_5	0x35	5 key
VK_6	0x36	6 key
VK_7	0x37	7 key
VK_8	0x38	8 key
VK_9	0x39	9 key
VK_A	0x41	A key
VK_B	0x42	B key
VK_C	0x43	C key
VK_D	0x44	D key
VK_E	0x45	E key
VK_F	0x46	F key
VK_G	0x47	G key
VK_H	0x48	H key
VK_I	0x49	I key
VK_J	0x4A	J key
VK_K	0x4B	K key
VK_L	0x4C	L key
VK_M	0x4D	M key
VK_N	0x4E	N key
VK_O	0x4F	O key
VK_P	0x50	P key
VK_Q	0x51	Q key
VK_R	0x52	R key
VK_S	0x53	S key

<b>Symbolic Name</b>	<b>Hexadecimal Value</b>	<b>Keyboard Equivalent</b>
VK_T	0x54	T key
VK_U	0x55	U key
VK_V	0x56	V key
VK_W	0x57	W key
VK_X	0x58	X key
VK_Y	0x59	Y key
VK_Z	0x5A	Z key
VK_NUMPAD0	0x60	Numeric keypad 0 key
VK_NUMPAD1	0x61	Numeric keypad 1 key
VK_NUMPAD2	0x62	Numeric keypad 2 key
VK_NUMPAD3	0x63	Numeric keypad 3 key
VK_NUMPAD4	0x64	Numeric keypad 4 key
VK_NUMPAD5	0x65	Numeric keypad 5 key
VK_NUMPAD6	0x66	Numeric keypad 6 key
VK_NUMPAD7	0x67	Numeric keypad 7 key
VK_NUMPAD8	0x68	Numeric keypad 8 key
VK_NUMPAD9	0x69	Numeric keypad 9 key
VK_MULTIPLY	0x6A	Multiply key
VK_ADD	0x6B	Add key
VK_SEPARATOR	0x6C	Separator key
VK_SUBTRACT	0x6D	Subtract key
VK_DECIMAL	0x6E	Decimal key
VK_DIVIDE	0x6F	Divide key
VK_F1	0x70	F1 key
VK_F2	0x71	F2 key
VK_F3	0x72	F3 key
VK_F4	0x73	F4 key
VK_F5	0x74	F5 key
VK_F6	0x75	F6 key
VK_F7	0x76	F7 key
VK_F8	0x77	F8 key
VK_F9	0x78	F9 key

<b>Symbolic Name</b>	<b>Hexadecimal Value</b>	<b>Keyboard Equivalent</b>
VK_F10	0x79	F10 key
VK_F11	0x7A	F11 key
VK_F12	0x7B	F12 key
VK_F13	0x7C	F13 key
VK_F14	0x7D	F14 key
VK_F15	0x7E	F15 key
VK_F16	0x7F	F16 key
VK_F17	0x80	F17 key
VK_F18	0x81	F18 key
VK_F19	0x82	F19 key
VK_F20	0x83	F20 key
VK_F21	0x84	F21 key
VK_F22	0x85	F22 key
VK_F23	0x86	F23 key
VK_F24	0x87	F24 key

## Glossary

### **CEBrowseX.IDACtl**

A Naurtech ActiveX control which provides access to the CETerm Automation Objects in the Windows Mobile PIE browser.

### **CETerm Automation Objects**

Built-in JavaScript objects that give access to CETerm, Device, and Operating System (OS) features. The objects are available within the browser page and the CETerm Scripting Engine. See the CETerm Scripting Guide for details.

### **external**

This is the name of an implied object in the DOM of the Windows CE IE6CE browser that gives access to the CETerm Automation Objects.

### **GetProperty**

A method available via the CETerm Automation Object to get a configuration value from the Naurtech Web Browser. See the CETerm Scripting Guide for documentation on GetProperty.

### **IDA Action Code**

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

### **IE6CE**

An acronym for “Internet Explorer 6 on Windows CE”. The Microsoft browser technology on Windows CE (non-Windows Mobile) devices. See Section 1.2 for details.

### **IEM6**

An acronym for “Internet Explorer Mobile 6”. The Microsoft browser technology on Windows Mobile 6.1.4 and later devices. See Section 1.2 for details.

### **PIE**

An acronym for “Pocket Internet Explorer”. The Microsoft browser technology available on all Windows Mobile devices. See Section 1.2 for details.

### **PostIDA**

A method available on the CETerm automation object to post (submit for deferred execution) an IDA Action Code to the Web Browser. See the CETerm Scripting Guide for documentation on PostIDA.

### **SendIDA**

A method available on the CETerm automation object to send (execute) an IDA Action Code to the Web Browser. See the CETerm Scripting Guide for documentation on SendIDA.

### **SetProperty**

A method available via the CETerm Automation Object to set a configuration value from the Naurtech Web Browser. See the CETerm Scripting Guide for documentation on SetProperty.

### **TextX (deprecated)**

A Naurtech ActiveX control that was used with Pocket PC or Windows Mobile 2003 platforms to provide a fully featured text input element which supported event handlers.

## Index

---

### *C*

CEBrowseX CLASSID · 60  
CEBrowseX Control · 59, 66

---

### *E*

external · 66

---

### *I*

iBrowse META tags · 18  
IDA Action Codes · 13  
IE6CE · 10  
IEM6 · 10

---

### *M*

META Tag Identifiers · 19  
  Application · 21  
  Battery · 21  
  BatteryNavigate · 22  
  Command · 24  
  CursorPos · 24  
  ErrorNavigate · 25  
  GetUnitInformation · 26  
  HomeKey · 27  
  IDA · 27  
  MoveSIP · 28  
  OnAllKeys · 29  
  OnKey · 30  
  PLSeriesLabel\_Complete · 42

PLSeriesLabel\_Print · 43  
PowerOn · 32  
Reboot · 32  
Scanner · 33  
ScannerNavigate · 34  
SetDate · 36  
SetTime · 36  
Signal · 37  
SignalNavigate · 38  
SIP · 39  
SIPUp · 40  
TextSize · 40  
TimerInterval · 41  
TimerNavigate · 41  
ZebraLabel\_Complete · 42  
ZebraLabel\_Print · 43

---

### *P*

PIE · 10  
PostIDA · 16, 66  
Print · 60  
PrintString · 60

---

### *S*

SendIDA · 67  
stiscan · 55  
stisubmit · 55

---

### *V*

Virtual Key Codes · 62