Cisco IP Phone BTXML Version 2.0 Application Development Guide

This document serves as a reference guide for developers who want to modify the user interface (UI) of Cisco IP phones equipped with BTXML support. Currently, BTXML is available only on the Cisco 7960 and 7940 IP phones with MGCP software.

Note Although the Cisco 7960 and 7940 IP phones are backward compatible with BTXML Version 1.0, Version 1.0 is obsolete. Anyone currently using Version 1.0 should migrate to Version 2.0.

This guide contains the following information:

About This Guide

This section discusses how to obtain additional documentation and technical assistance.

Obtaining Documentation

These sections explain how to obtain documentation from Cisco Systems.

World Wide Web

You can access the most current Cisco documentation on the World Wide Web at this URL:

http://www.cisco.com

Translated documentation is available at this URL:

http://www.cisco.com/public/countries_languages.shtml

Documentation CD-ROM

Cisco documentation and additional literature are available in a Cisco Documentation CD-ROM package, which is shipped with your product. The Documentation CD-ROM is updated monthly and may be more current than printed documentation. The CD-ROM package is available as a single unit or through an annual subscription.

Ordering Documentation

You can order Cisco documentation in these ways:

- Registered Cisco.com users (Cisco direct customers) can order Cisco product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/go/subscription

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco Systems Corporate Headquarters (California, U.S.A.) at 408 526-7208 or, elsewhere in North America, by calling 800 553-NETS (6387).

Documentation Feedback

You can submit comments electronically on Cisco.com. In the Cisco Documentation home page, click the **Fax** or **Email** option in the “Leave Feedback” section at the bottom of the page.

You can e-mail your comments to bug-doc@cisco.com.

You can submit your comments by mail by using the response card behind the front cover of your document or by writing to the following address:

Cisco Systems
Attn: Document Resource Connection
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides Cisco.com as a starting point for all technical assistance. Customers and partners can obtain online documentation, troubleshooting tips, and sample configurations from online tools by using the Cisco Technical Assistance Center (TAC) Web Site. Cisco.com registered users have complete access to the technical support resources on the Cisco TAC Web Site.

Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information, networking solutions, services, programs, and resources at any time, from anywhere in the world.

Cisco.com is a highly integrated Internet application and a powerful, easy-to-use tool that provides a broad range of features and services to help you with these tasks:

- Streamline business processes and improve productivity
- Resolve technical issues with online support
- Download and test software packages
- Order Cisco learning materials and merchandise
- Register for online skill assessment, training, and certification programs

If you want to obtain customized information and service, you can self-register on Cisco.com. To access Cisco.com, go to this URL:

http://www.cisco.com

Technical Assistance Center

The Cisco Technical Assistance Center (TAC) is available to all customers who need technical assistance with a Cisco product, technology, or solution. Two levels of support are available: the Cisco TAC Web Site and the Cisco TAC Escalation Center.

Cisco TAC inquiries are categorized according to the urgency of the issue:

- Priority level 4 (P4)—You need information or assistance concerning Cisco product capabilities, product installation, or basic product configuration.
- Priority level 3 (P3)—Your network performance is degraded. Network functionality is noticeably impaired, but most business operations continue.
- Priority level 2 (P2)—Your production network is severely degraded, affecting significant aspects of business operations. No workaround is available.
- Priority level 1 (P1)—Your production network is down, and a critical impact to business operations will occur if service is not restored quickly. No workaround is available.

The Cisco TAC resource that you choose is based on the priority of the problem and the conditions of service contracts, when applicable.

Cisco TAC Web Site

You can use the Cisco TAC Web Site to resolve P3 and P4 issues yourself, saving both cost and time. The site provides around-the-clock access to online tools, knowledge bases, and software. To access the Cisco TAC Web Site, go to this URL:

http://www.cisco.com/tac

All customers, partners, and resellers who have a valid Cisco service contract have complete access to the technical support resources on the Cisco TAC Web Site. The Cisco TAC Web Site requires a Cisco.com login ID and password. If you have a valid service contract but do not have a login ID or password, go to this URL to register:

http://www.cisco.com/register/

If you are a Cisco.com registered user, and you cannot resolve your technical issues by using the Cisco TAC Web Site, you can open a case online by using the TAC Case Open tool at this URL:

http://www.cisco.com/tac/caseopen

If you have Internet access, we recommend that you open P3 and P4 cases through the Cisco TAC Web Site.

Cisco TAC Escalation Center

The Cisco TAC Escalation Center addresses priority level 1 or priority level 2 issues. These classifications are assigned when severe network degradation significantly impacts business operations. When you contact the TAC Escalation Center with a P1 or P2 problem, a Cisco TAC engineer automatically opens a case.

To obtain a directory of toll-free Cisco TAC telephone numbers for your country, go to this URL:

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

Before calling, please check with your network operations center to determine the level of Cisco support services to which your company is entitled: for example, SMARTnet, SMARTnet Onsite, or Network Supported Accounts (NSA). When you call the center, please have available your service agreement number and your product serial number.

Introduction to BTXML

What Is BTXML?

BTXML (Basic Telephony Extended Markup Language) is a scripting language and system that allows for enhanced control of IP phones. It provides an application infrastructure for IP telephony, independent of protocol and terminal.

This infrastructure provides the following benefits:

- Allows Cisco to deploy features more quickly.
- Provides an abstract application programming interface (API), which allows applications to work on multiple phone types.
- Allows customers to create custom features and services independent of Cisco releases.

BTXML is based on the eXtensible Markup Language (XML) standard but is tailored to the needs of telephone handsets.

Note The BTXML system includes a translator module to maintain compatibility with applications written for Cisco Call Manager XML (CMXML) Version 3.0 and earlier versions.

What BTXML Can Do

The core of the BTXML system is an XML-based microbrowser that has support for text, graphics, and input processing. The microbrowser processes all user interaction through *Cards*, which contain BTXML language commands. This allows the user interface to be tailored to match the user's needs.

The Microbrowser

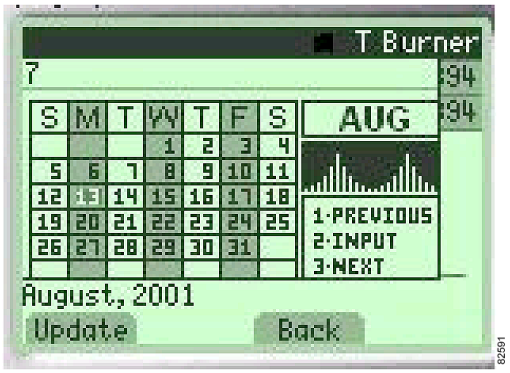
The idea behind using a microbrowser on a telephone is nothing new. Wireless Application Protocol (WAP)-based phones, which require a gateway to translate special web pages, have minibrowsers. The BTXML browser goes beyond this usage to provide integrated telephony features. With BTXML, a much higher degree of interactivity and formatting control is available, as shown in the following application sample screens.

Figure 1 shows how BTXML can be used to display a list of custom services for a series of phones from a common server.

Figure 1 BTXML Services Screen

When the user selects one of these services (either by typing the item number or by highlighting an item and using the Select softkey), BTXML displays the selected application. For example, if item 4 is selected, the phone links to the calendar application as shown in Figure 2.

Figure 2 The Calendar Application



The BTXML microbrowser supports the following functionality:

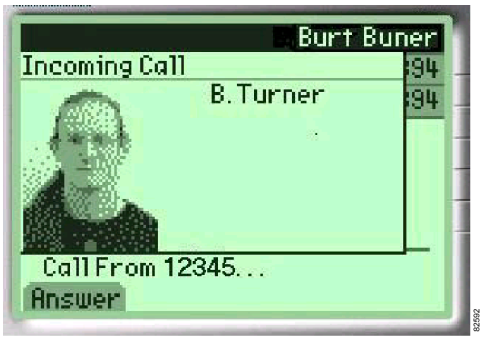
- Multiple input fields with the same card
- Graphic images with text wrapping
- Input-key mapping
- Dynamic soft key assignments
- Arbitrary scaling and dithering of standard Windows bitmap (.BMP) files

Note Graphic images must be in Microsoft Windows 256-color BMP format. Other formats such as GIF and JPEG are not supported.

- Special function widgets, like the Slider, to adjust system parameters

This functionality leads to a wide variety of applications that can interact with the phone. Figure 3 shows an application that could display a bit-mapped representation of the calling party on an incoming call.

Figure 3 Incoming Call Bitmap



What BTXML Cannot Do

BTXML is not a general-purpose programming language. It cannot be used for the following:

- mathematically intense tasks
- sorting large amounts of data in real time
- performing call-control functions on the phone

BTXML Overview

To maintain simplicity, BTXML has only a few elements, as described in the following sections:

Structural Elements

The following are BTXML structural elements:

- **btxml**—collection of decks
- **deck**—collection of cards
- **card**—basic unit of an application state (contains all other elements)
- **status**—text/icons for the status line
- **window**—rendered sub-area of the display
- **background**—sets the background area of the display
- **keybindings**—associates a key with an action
- **topbar**—sets the topbar portion of the screen

Component Elements

The following are BTXML component elements:

- **title**—specifies a title string for a window
- **item**—identifies an element in a list
- **icon**—inserts a graphical representation character
- **image**—inserts a bitmap picture
- **calltimer**—inserts a timer showing the length of the current call
- **font**—changes the color of the rendered text
- **br**—inserts a line break
- **rj**—causes subsequent text to be right justified
- **slider**—inserts a slider
- **timer**—inserts a running clock time

Control Elements

The following is a BTXML control element:

- **timer**—specifies a time-out event.

Modifier Elements

The following are BTXML modifier elements:

- **update** —update existing items with new values
- **set**—sets a variable to a new value

XML Rules

Some rules to remember regarding XML are:

- Element names are case sensitive.
- The text of any attribute is stripped of characters below 0x20 in value (the space character).
- Text in the content portion of the element is *not* stripped.
- XML extended escape sequence for characters may be used, such as *{* or *«*.
- XML escapes are also valid: *<*, *>*, *"*, *&*, * *, *'*.
- XML comments of the form *<!-- Comment -->* are supported.
- The XML parser has a maximum predefined limit of 1,024 characters for each post.

BTXML Syntax

This section describes the syntax used in BTXML cards. Before looking at the syntax details, we will look at the general format of a BTXML file. In the example below, notice that BTXML syntax looks a lot like standard XML 1.0 syntax, which in turn looks very much like HTML.

Note BTXML is not entirely XML 1.0 compatible. BTXML differs in the following areas:

- Support for nondefined entities, such as BTXML variables
- No recursive support for entities
- No support for DTD syntax

The program or script is made up of *tokens*, which are enclosed in brackets < >, and text strings, which specify attributes for the various tokens. Nesting (indenting) may optionally be used to make the program more readable, but the BTXML parser does not require indenting.

All tokens follow a similar pattern:

- <token *attributes*>
- content
- </token>

The opening <*token*> phrase establishes what token or keyword you want to use. The attributes modify the token or provide additional details. The *content* would normally be a text string or other data the token can use. The whole script is closed with the </*token*> phrase. See Example 1.

Example 1 Example BTXML file

```
<?xml version="1.0" ?>
<btxml>
<deck id="examp">
<card id="exampcard">
    <timer value="0" />
    <keybindings>
        <item id="EXAMP_KEY" onpick="event=EXAMP_TEST">Test</item>
    </keybindings>
    <window>
        This is some text
    </window>
    <status>
        Please press the Test key
    </status>
</card>
<card id="nextcard">
    <!-- Your card contents here -->
</card>
</deck>
</btxml>
```

BTXML Specifications

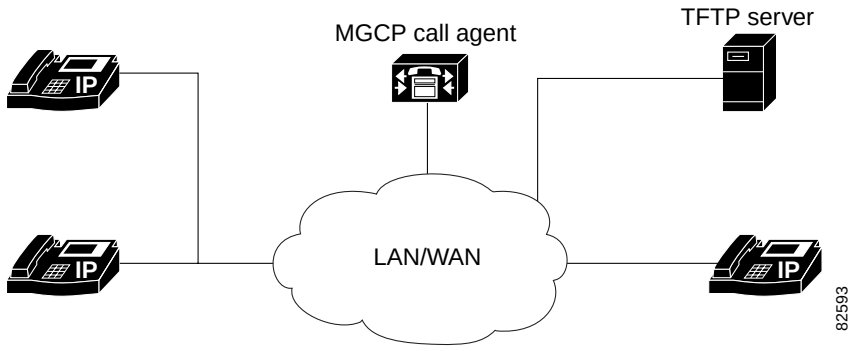
This section provides a reference for the graphic elements used in a BTXML application and the associated syntax.

BTXML Environment

This section explains the physical environment in which BTXML runs.

The BTXML environment consists of telephone handsets, servers, and the network that connects them. The handsets consist of a GUI display and a keypad. See Figure 4.

Figure 4 BTXML Environment



GUI Display

The basic GUI display consists of three *planes* or levels.

- Background—contains the background browser, the top bar, the status bar, and the soft keys
- Middle—contains the line keys
- Front—contains the browser window

Each layer overlays the preceding layer. This means that anything that appears in the middle plane overwrites anything in the corresponding area of the background plane. This simplifies interaction with the phone and guarantees a standard interface across all phone platforms.

Figure 5 shows a basic telephone screen with the some of the basic display elements in place.

Figure 5 Basic Telephone Screen

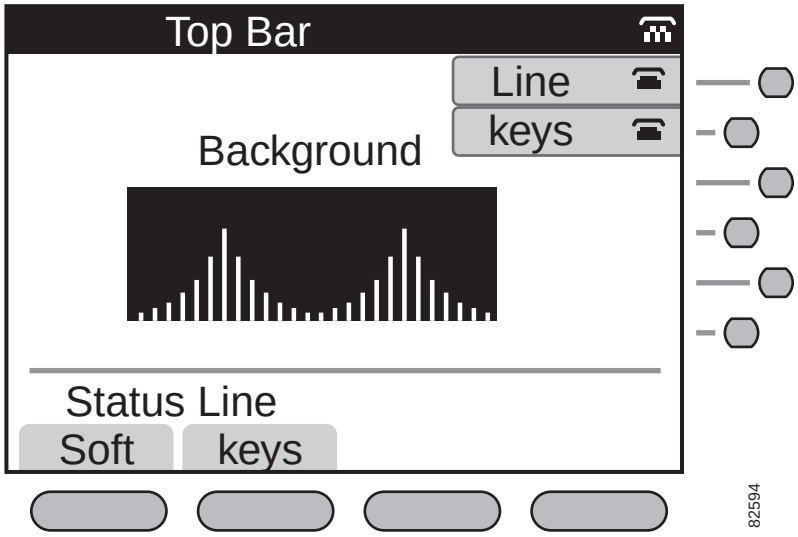
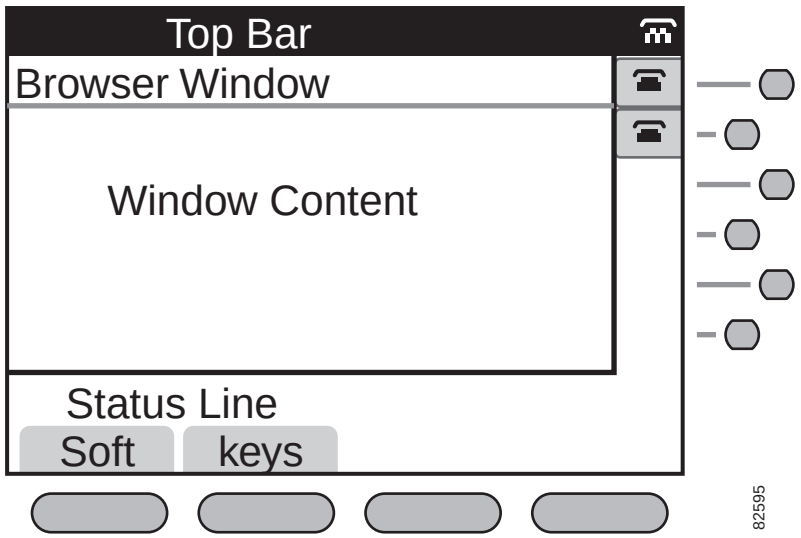


Figure 6 shows the same screen with the browser window open. Note how the content of the browser overlays the line keys and the background. When the browser window is closed, the line keys and the background are restored automatically.

Figure 6 Screen with Browser Window Open



- Graphic elements—The microbrowser window can contain any combination of text, icons, menus, sliders, and so on, with certain limitations. This allows for a great deal of freedom when designing applications.
- Item numbers—As the browser parses the page, it assigns item numbers to each element in the order in which they are encountered. The item numbers are used to hot-select the item from anywhere on the page. When a menu is displayed, for example, each menu item has a number appended to it on the display. The user can either scroll to the item using arrow keys or simply type the number in using the numeric keypad.

The Keypad

The keypad allows the user to input data and events into the telephone. A typical Cisco 7960 IP phone is shown in Figure 7.

Figure 7 Cisco 7960 IP Phone

Note Although the Cisco 7960 IP phone is used as an example, most of the information below applies to the Cisco 7940 IP phone.

Although the phones may have different keypads, several standard keys are common to all:

- Alphanumeric keypad—These keys are used for data entry. Depending on the input mode (numeric or alphanumeric), each key can either be used to enter a number (0–9, *, and #) or an alphabetic character. Typically, when input is required, a softkey allows the user to toggle between the two modes
- Vertical navigation keys—The navigation key allows scrolling through the browser. If no items are visible, the browser scrolls one line at a time. If items are present, the browser selects the first item, and then further scrolling causes successive items to become highlighted.
- Horizontal navigation keys—These keys are typically used to adjust volume and other similar controls. Normally, a slider widget is present on the screen when these keys are active.
- Speaker, headset, and mute—These keys are not usually associated with the browser pages, but can be interpreted by BTXML if desired.
- Directory—If a directory key is present, it is typically either tied to the built-in personal directory system, or to a custom LDAP adapter to access corporate directories.

- Services—The services key causes the phone to access whatever applications the administrator wants to install on the application server, if any. This may include the Cisco Call Manager services applications, but can also be completely custom applications created for a specific system.
- Info key—The info key,"i", is used to allow the user to get help for individual keys.
- Messages—If a messages key is present on the phone, it is usually configured by the administrator to access the company voice messaging service.
- Settings—The settings key allows the user to view or change various options on the phone.
- Softkeys—The softkeys are software-programmable keys that change labels and functions depending on context. The values and functions of these keys are completely programmable via the XML cards. If the number of softkeys exceeds four, BTXML automatically divides the keys into groups of three and provides a "more" key to allow movement between groups.
- Line keys—Each telephone line has one line key associated with it. These can either be used directly by the program or can be overlaid with special softkeys for customized functions.

Call Agent

Each MGCP network has a call agent that controls a number of associated phones. The call agent may communicate with the phone’s BTXML system using MGCP request and notify messages.

HTTP Server

A network may have one or more application servers that host user applications. A user application can, for example, be used to format a list of sales contacts and route the resulting list to a specific phone display. The phone communicates with the HTTP server using the Cisco Call Manager 3.0 scripting language and is typically accessed through the Services or Directory keys.

In smaller networks, the application server can be hosted on the same physical computer as the TFTP server.

For more information on the Cisco Call Manager 3.0 scripting language, refer to the following links:

- Cisco IP Phone Service forum

  www.hotdispatch.com/cisco-ip-telephony

- Cisco Call Manager SDK

  http://www.cisco.com/warp/public/570/avvid/voice_ip/cm_xml/cm_xmldown.shtml

Also refer to the book *Developing Cisco IP Phone Services*, by Darrick Deel, Mark Nelson, and Anne Smith, ISBN 1-58705-060-9.

BTXML Entities

This section discusses the basic entities that are supported in the BTXML language.

Tip These terms are used throughout much of the text that follows. An *Element* is a complex object that exists on the screen, such as a menu. A *widget* is a lower-level entity that exists as part of an element, such as an icon or timer.

The standard BTXML screen contains a combination of the following elements and widgets:

- Top bar—this displays the phone label, and indicator of the active protocol, and possibly the current date and time.
- Status line—shows current notification messages of events and potential actions. On smaller displays, the status line is often incorporated into other windows. The status line is generally as wide as the total screen width.
- Soft keys—custom action labels specific to the current phone state. Each softkey corresponds to a push button at the bottom of the screen. If the BTXML card specifies more than four softkeys at one time, the system automatically breaks the softkeys into groups. Each group consists of three softkeys, with a "more" key in the fourth position.
- Line keys—for multiline phones, the line key plane is commonly kept on screen at all times and shows the line state and action associated with the line key. Line keys are generally located on the right side of the display and consist of a physical push button, and a logical display element on the display. The display element normally consists of a text-line name and an icon showing the line state. These display elements are enclosed in a lozenge-shaped outline.
- Background browser—displays the default idle screen and a backdrop for all other windows that pop in front of it. The background may optionally contain a graphic display, such as a corporate logo, which can be downloaded as a bitmap.
- Browser window—the Browser window displays whatever content is specified in the BTXML cards.
- Icon or animated icon—an icon is a widget that can be included in any display element. A number of icons are included in the phone software and can be accessed by specifying the icon keyword and the icon ID. Animated icons consist of a group of icons that can be sequenced to produce animation on the screen.
- Image—an image is a bitmapped image, such as a logo, that can appear on a card. Images are sent to the phone in the Microsoft Windows uncompressed 256-color bitmap (BMP) format.
- Input item—this item is used whenever the application needs to ask for data from a user. An example of this is the basic dialing capability of the phone. When the user wishes to initiate a call, an input item is generated to gather the keypress information. The input item can handle numeric or alphanumeric information as well as certain special characters, such as /, \, :, and so on.
- Menu item—the menu item element is the basis for building menus on the screen. Each menu item contains a label and is linked to specific actions that occur when the item is selected.
- Slider bar—the slider is a graphic element that emulates a control knob. An example is the contrast control on the phone. The value of the slider is transferred to the application through a simple numeric variable specified in the BTXML card.
- Call timer—the call timer displays time in a standard HH:MM:SS format. Whenever a call is active, the associated call timer appears on the screen.

- Card—a BTXML card contains BTXML code. When a card is sent to the BTXML parser, each statement on the card is parsed and action is taken based on the enclosed statements. A card can describe a new menu, a set of softkeys, a graphic bitmap or any other BTXML element, or a combination of elements.
- Deck—a deck is a collection of cards. Typically, a deck is created to support an application and contains cards corresponding to each possible action that may happen within the application. Decks can be downloaded into the phone, but each phone also contains one or more built-in decks to support standard functionality.
- Application—there are two application types: internal and external. Internal applications are written in BTXML and can be modified using the MGCP request/notify mechanism. External applications are hosted on an HTTP server and communicate with the phone using the Cisco Call Manager XML, Version 3.0.
- HTTP application server—application servers are programs written in a high-level language, such as PERL or Java, that exist on a computer. Typically, an application server communicates with the phone through HTTP messages and can connect the phone with databases or other applications. An LDAP interface server is a good example of this type of functionality.

BTXML Elements

This section discusses the basic elements that are supported in the BTXML language.

btxml

The **btxml** keyword encapsulates a deck of one or more cards, as in the following example:

```
<btxml>
    <!-- Deck goes here -->
</btxml>
```

deck

The **deck** keyword encapsulates one or more cards, as in the following example:

```
<deck id="deckid">
    <!-- Deck content goes here -->
</deck>
```

The *id* attribute names the deck and must be unique within the script. See Table 6 for a list of containment rules for this element.

card

The **card** keyword encapsulates a single card, as in the example that follows:

```
<card id="cardid">
    <!-- Card content goes here -->
</card>
```

The *id* attribute names the card and must be unique within the card deck. Any elements that are specified in the card replace elements that are currently on screen. For example, if a <*status*> element is included in the card, the current status area is overwritten with the new content. See Table 6 for a list of containment rules for this element.

status

The **status** keyword is used to display a message in the status area of the screen, as in the following example:

```
<card id=”prompter”>
    <status>Your Current Options</status>
</card>
```

This keyword is typically used to prompt the user for information or to display an error or status message. See Table 6 for a list of containment rules for this element.

window

The **window** keyword is used to build a display window composed of one or more other elements, as in the following example:

```
<window width="wide|narrow"
    onhilite="href"
    href="href">
    <!-- Window content goes here -->
</window>
```

The **window** keyword has the following attributes:

- width—specifies the width of the window which may be either wide or narrow (default). A narrow window allows the information from the line keys on the right side of the screen to be seen.
- onhilite—if this optional attribute is present, the supplied HREF acts as the default hilite for the window. This means that the HREF will be executed whenever the screen is scrolled, but there are no items on the screen that can be highlighted. This can be used to remove softkeys from the display that are not currently valid.
- href—if this optional attribute is present, the content of the <window> element is ignored. Instead, HREF value is executed and the resulting page is loaded into the window.

See Table 6 for a list of containment rules for this element.

background

The **background** keyword is used to control the content of the background plan, as in the following example:

```
<background href="href">
    <!-- Background content goes here -->
</background>
```

The background keyword has the following attribute:

- href—If this optional attribute is present, the content of the <background> element is ignored. Instead the HREF is executed and the resulting page is loaded into the background.

See Table 6 for a list of containment rules for this element.

keybindings

The **keybindings** keyword is used to bind individual keys to corresponding actions, as in the following example:

```
<keybindings>
    <!-- Keybinding Items go here -->
    <item id="asdf" onpick="href" flags=”flags”>Name</item>
</keybindings>
```

When a new card is loaded into the system, some or all of the keys on the phone may be changed to perform new functions.

The **keybindings** keyword has the following attributes:

- id—Identifier for this key (see Tables 1 through 3 for more details).
- onpick—Action taken when this key is pressed.
- flags—Optional modifiers for this key.

Keybindings may be divided into a few basic classes:

- hard-keybindings—Keys that are hard-coded on the phone, such as line keys or navigation buttons.
- soft-keybindings—Keys that display context-sensitive labels on the display screen.
- sticky-keybindings—Keys that function no matter what application is currently running.
- nonsticky-keybindings—Keys that function only when their own application is running.

Key types may be freely intermixed within a card.

Tip A key may be designated as sticky by setting its flag field to "S."

When a keybinding element is encountered and processed within a card, all previous nonsticky keybindings (from other cards) are deleted before the new keys take effect. If a keybinding is encountered for a key that was previously a sticky-key, the sticky-key binding is removed and the new binding takes effect.

Tip On phones with line keys, unused lines may be set up as soft keys by creating a softkey with the sticky flag set.

The predefined hard-key IDs available through BTXML are listed below. Any other key used by the application that is not recognized is automatically placed on the list of softkeys.

Keys are split into three subgroups (see Table 1 to Table 3):

- Tier 1 (Standard)—Hard keys available on all phones.
- Tier 2 (Common)—Most phones have these keys.
- Tier 3 (Platform)—Keys that are particular to a specific platform.

Table 1 Standard Key IDs (Tier 1)

Key	Description
KEY_0 - KEY_9	Standard number pad keys
KEY_POUND	# key
KEY_STAR	* key
KEY_OFFHOOK	Handset out of cradle
KEY_ONHOOK	Handset in cradle
KEY_FLASH	Fast onhook/offhook transition
KEY_HALFHOOK	This is sent when the hook switch is initially pressed to distinguish between an onhook and a flash event, yet still allow the user interface to respond to the user event.

Table 2 Common Key IDs (Tier 2)

Key	Description
KEY_HEADSET	Headset on/off key
KEY_SPEAKER	Speaker on/off key
KEY_MUTE	Mute key
KEY_VOL_UP	Volume up key
KEY_VOL_DOWN	Volume down key
KEY_UP	Up navigation key
KEY_DOWN	Down navigation key
KEY_ACCEPT	Accept key
KEY_CANCEL	Cancel key

Table 3 Platform Specific Keys, Cisco 7960/7940 (Tier 3)

Key	Description
KEY_INFO	I key
KEY_DIRECTORY	Directory key
KEY_MESSAGES	Messages key
KEY_SERVICES	Services key
KEY_SETTINGS	Settings Key

To understand the keybindings keyword more fully, it is helpful to examine a few basic examples.

The following example is a simple card that opens a wide window and fills it with the image found in the file im1.bmp:

```
<card id=”thiscard”>
    <window width="wide">
        <image href="http://server/dir/images/im1.bmp" />
    </window>
    <keybindings>
        <item id="Ask" onpick="event=EVENT_ASK">Ask?</item>
    </keybindings>
</card>
```

The card then creates a softkey on the display. The softkey has the *ASK?* label visible and posts an *EVENT_ASK* event whenever it is pressed. The *EVENT_ASK* event is a user-defined event sent to the application, causing an action to take place.

Note Although the image tag could be implemented as <image></image>, it is also acceptable to abbreviate the closing tag in many cases by using the ‘/>’ token. Using this method, one could shorten syntax, such as `<image href=""> </image>, to the more compact <image href="" />.`

The use of an empty content section when defining softkeys has special significance. It tells BTXML that you would like to skip (leave blank) one key on the screen. In the next example, two softkeys are defined with an empty space between them:

```
<card id=”thiscard”>
    <window width="wide">
        <image href="http://server/dir/images/im1.bmp" />
    </window>
    <keybindings>
        <item id="Ask" onpick="event=EVENT_ASK">Ask?</item>
        <item id=”Foo” onpick=”null”></item>
        <item id=”Exit” onpick=”event=APP_RELEASE”>Exit</item>
    </keybindings>
</card>
```

Notice that the second key, Foo, has no content defined after the initial *<item>* token. A blank space is, therefore, left on the second softkey location. If the user presses this key, no action is taken.

Note When a user presses the "more" key to select the next group of softkeys, a five-second timer is started. If no softkey is pressed before the time expires, the softkey set automatically reverts back to the first three keys.

Tip When designing your application, always try to arrange softkeys and menus according to projected frequency of use. The most common keys and menu items should appear earliest in the list. Least used keys and items should appear last.

One other special use of keys involves the creation of on-screen menus. In the next example, a very simple menu is created:

```
<card id=”MYMENU”>
    <window width="wide">
        <title>My First Menu</title>
        <item type=”menu” onpick=”event=THIS”>This</item>
        <item type=”menu” onpick=”event=THAT”>That</item>
        <item type=”menu” onpick=”event=THEOTHER”>Other</item>
    </window>
    <keybindings>
        <item id="KEY_ACCEPT" onpick="event=KEY_ACCEPT">Select</item>
        <item id="NULL" onpick="null" />
        <item id="KEY_CANCEL" onpick="event=GO_BACK">Back</item>
    </keybindings>
</card>
```

When this card is invoked, a three-item menu is displayed. The menu items are *this*, *that*, and *other*. The user can choose one of these items in two ways. The first is to simply press the 1, 2, or 3 key. Doing this sends the associated event to BTXML.

The second way to access an item is to use the navigation (scroll) keys, which causes each item to be highlighted in turn. When the desired item is highlighted, the user can press the “select” softkey. When this is done, BTXML finds the highlighted key and sends the associated onpick event to the application. The *key_accept* event is not seen by the application.

title

The **title** keyword allows you to define a title for the current window, as in the following example:

```
<window width=”wide|narrow">
    <title><icon id=”INFO”>Information</title>
</window>
```

Notice that the icon with the *INFO* identifier is displayed to the left of the title text, Information. See Table 6 for a list of containment rules for this element.

item

The **item** keyword is used to define any user interface (UI) component that receives user input and responds accordingly, as in the following examples:

```
<item type="menu|input"
    id="keyID"
    name="varName"
    value="varValue"
    onhilite="href"
    onpick="href"
    flags="flags">TEXT of Item
</item>

<item id="KEY_ACCEPT" onpick="event=KEY_ACCEPT">Select</item>
<item type=”menu” onpick=”event=THIS”>This</item>
```

Items include menu items, key bindings, and input fields. See Table 6 for a list of containment rules for this element.

The attributes for the **item** keyword change according to the context in which the item is being used. See Table 3. The descriptions of each item follow:

- type—Sets the type of window item (menu or input).
- id—ID of item for later use by <update>.
- content—Text string displayed with this item.

Note You do not need to actually type the word *content* when defining the content attribute. The text string is recognized as the content automatically, purely from the context in which it is found.

- name—Name of variable to add to the application context when this item is picked.
- value—Value of named variable. This value is set each time a digit or character is entered into the string.
- onhilite—Executed whenever this item is highlighted. Commonly used to modify softkeys associated with this item.
- onpick—Executed whenever this item is picked by the user. An item is picked when it is highlighted and the user causes the *key_accept* event to occur, or when the user selects the number of the item.
- flags—Flags that further describe an individual item.

Table 4 Item Attributes for Specific items

Item Type	Attributes

Menu Items
Note A maximum of 32 menu items may be defined within a given card.
	• type—Required (type = “menu”)
	• content—Used as text for menu
	• id—Not required
	• onhilite—Executed when hilighted
	• onpick—Executed when picked
	• flags
	  • “-”—Do not autoexecute when selected
	  • “U”—Do not show menu item number

Input Items	• type—Required (type = “input”)
	• content—Ignored
	• name—Required
	• value—Used as default entry for the input.
	• flags:
	  • “A”—Allow alpha characters
	  • “N”—Allow digits
	  • “*”—Password. After entering a character, it is displayed as “*.”
	  • “D”—Dial plan. Pass the value through the dial plan system before using.
	  • “E”—Equation. Allow basic math equations to be formed from the symbols 0-9 ( ) . + - ^ * and /.

Keybinding Items **Note** Keybindings Items *must* be enclosed within a <keybindings> block. **Note** A total of fifteen keys can be bound within a keybinding block, with a maximum of six line keys, and one binding for each hardkey.	• type—Ignored • id—Name of the key to bind to, or unique name for a soft key. • content—Ignored for hard keys. Content contains label to display on screen for soft keys. • name—Ignored • value—Ignored • flags: • “S”—Key is sticky; not removed when new keybindings are set, unless the new keybinding overlays the same key ID.

icon

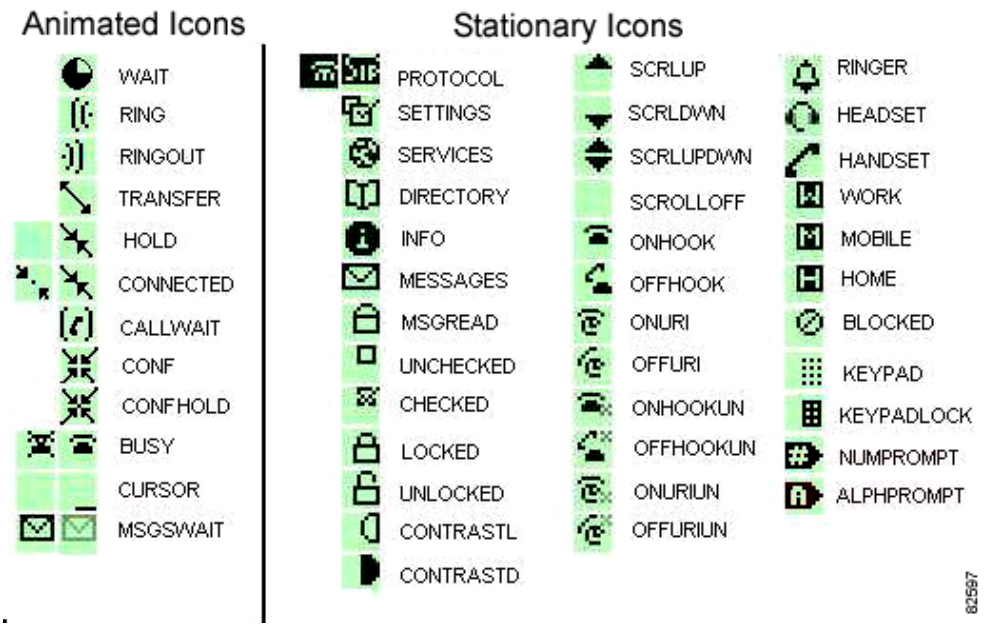
The **icon** keyword is used to display a graphic icon on the screen, as in the following example:

```
<icon id="icon_name"/>
```

Note Although the icon tag can be implemented as <icon></icon>, it is also acceptable to abbreviate the closing tag in many cases by using the '/>' token. Using this method, you can shorten syntax, such as <icon id=icon_name> </icon>, to the more compact <icon id=icon_name/>.

Figure 8 shows the icons that can be used in the phone. Icons are indexed by using the text name shown next to each one.

Figure 8 Icon IDs



image

The **image** keyword is used to display a graphic bitmap on the screen, as in the following example:

```
<image height="#pixels"
       width="#pixels"
       href="image_href">
</image>
```

This command causes an image to be displayed on the phone, scaled the size indicated by height and width parameters.

Caution Although BTXML dynamically scales an image, it scales only in the downward direction. An image that contains 10x10 pixels is *not* scaled upward to 100x100 pixels.

Note BTXML expects images to be formatted as Microsoft Windows (BMP) 256-color, uncompressed format. Any other format may cause unpredictable results, such as the image showing up as a blank on the screen.

Note If the width or height of the image (as specified in the BTXML code) exceeds the screen size, the image is scaled down to fit the screen automatically.

calltimer

The **calltimer** keyword is used to display a running timer for a call, as in the following examples:

```
Syntax:
<window>
    <calltimer value="#seconds" callid="unique_id"/>
</window>

Example:
<window>
    <calltimer value="&timer2;" callid="&id;"/>
</window>

Example:
<window>
    <calltimer value="03:21:00" callid="&id;"/>
</window>
```

The value attribute defaults to 0 and is the value that is displayed in seconds in the call timer field.

The *callid* attribute is a string that identifies the call and must be unique if more than one calltimers are displayed at the same time.

In the first coding example above, variables are being used (&id) to pass application variables into the card. See variables (&x), page 28, for more information on this technique.

In the second example above, an absolute time value is passed into the timer in the HH:MM:SS format. One can also pass a time into the timer formatted as a number of seconds, such as “23122”. No matter which format is passed in to *set* the timer, the time is always displayed in HH:MM:SS format.

Note The timer on the Cisco 7940/7960 IP phones resets to 0 after 100 hours.

font

The **font** keyword is used to adjust the appearance of text on the display, as in the following examples:

```
Syntax:
<window>
    <font fore="WHITE|BLACK|LTGREY|DKGREY"
           back="WHITE|BLACK|LTGREY|DKGREY">
    Text to be displayed
    </font>
</window>

Example:
<window>
    <font fore="DKGREY" back="LTGREY"> Hello World! </font>
</window>
```

See Table 6 for a list of containment rules for this element.

br

The **br** keyword forces a line break in the text on the display, as in the following example:

```
<window>
    This text is on one line
    <br/> This text on another line
</window>
```

Note Although tags such as br and rj could be implemented as
</br>, the abbreviated form <br/> is preferred. It is also acceptable to abbreviate the closing tag on many other cases by using the '/>' token. Using this method, one could shorten syntax such as <icon id=icon_name> </icon> to the more compact <icon id=icon_name/>.

rj

The **rj** keyword changes the default justification so that all text after the tag is right justified, as in the following example:

```
<window>
    This text left justified
    <rj/> This text is right justified
    <br/> This is left justified
</window>
```

This applies to text and icons only. The right justification lasts until any of the following situations occur:

- The window is closed with the </window> tag.
- The line is broken with a <br/> tag.
- The current text line exceeds the length of the display.

Text normally defaults to left justification.

set

The **set** keyword creates a permanent variable in the current application's context, as in the following example:

```
<window>
    <set name="shift" value="alpha">
</window>
```

The attributes for this tag are:

- name—The name of the variable to create.
- value—The value to be used for the new variable.

timer

The **timer** keyword creates a timer that counts down a specified number of seconds, then executes the specified href, as in the following example:

```
<card id = “mycard”>
    <timer value=”2” href=”event=APP_RELEASE”> </timer>
    <window>
        This is a test
    </window>
</card>
```

The time can be canceled by posting a new timer with the value set to 0.

The attributes for this tag are:

- value—The number of seconds to count.
- href—What to do when the timer expires.

slider

The **slider** keyword creates an on-screen slider that displays an analog value from 0 to 100, as in the following example:

```
<status>
    <slider id=”SLIDE1” value=”&slideval;” />
</status>
```

The attributes for this tag are:

- id—A unique id for this slider
- value—A variable that is used to pass the value to and from the slider

update

The **update** keyword allows the user to modify the attributes and content of existing elements, as in the following example:

```
<update>
    <item id=”olditemid”
        onpick=”newonpick”
        onhilite=”newonhilite”
        name=”newname”
        value=”newvalue”>New Content </item>
</update>
```

Also, see Table 6 for a list of containment rules for this element.

<?xml> (Card Update)

While the **update** keyword is useful for updating existing elements, the *Card Update* token <?> allows you to replace existing cards in the phone.

When card decks are sent to the phone, they are loaded into memory for later reference. At times, it might be convenient to change some of the existing cards. This is what the <?> token allows you to do.

For example, assume that there is a deck named *basic* in a phone and in this deck there is a card called *VoiceMail*. If we want to change that card, we can do it with the syntax listed in the following example:

```
<?xml version=”1.0” ?>
<btxml>
    <deck id=”basic”>
        <card=”VoiceMail”>
            <!-- New contents go here -->
        </card>
    </deck>
</btxml>
```

When the cards have been updated, the phone always executes the card called *load* in the current application deck. The *load* card performs the necessary actions to utilize the new cards.

variables (&x)

Although they are not tokens as such, it is important to mention variables as part of the basic syntax of BTXML. Almost all of the tokens in BTXML accept static text strings as arguments or attributes. These variables allow the card to be changed dynamically by the current application.

Variables have the general format of *&var*, where *var* is the variable name. When the BTXML parser processes a card with a variable in it, the parser automatically substitutes the variable value into the card before rendering it on the display.

In the following example, the variable *volume_value* is used to set the slider to a given position:

```
<card id="MyVolumeCard">
    <status>
        <icon id="LTSPACE16" />
        <slider id="Volume1" value="&volume_value" />
    </status>
</card>
```

Whenever the application wants to change the position of the slider thumb, it reposts the card with a new value.

Tip Although the basic syntax of a variable is relatively simple, there is more to it than appears in the example above. The following section discusses the proper use of the ampersand (&) character.

The Truth About Ampersand &

If anything is likely to confuse a new BTXML programmer, it is the use of the ampersand (&) character. The following discussion helps explain the proper use of this character and its close relative, the *&amp* token.

At its lowest level, the BTXML parser is very adept at processing lists of variables, which are sent to it as name/value pairs. Each of these pairs uses the form *name = value*, which is the standard syntax for HTML. If only a single variable were allowed, things would be very simple for the parser. Unfortunately, the parser is often called upon to process multiple name/value pairs within one request. To keep these pairs separated, a delimiter is required and based upon the precedence set in HTML, that delimiter is the ampersand.

The parser expects values in a list of the format *name1=value1&name2=value2*, and so on. If the ampersand were not present, *value1* would blend into *name2*, leading to confusion for the parser.

BTXML also uses the ampersand character to identify variables within cards. The dual use of this character can, therefore, lead to a some confusion for the parser. If, for example, we pass the list *name1=value&name2=value2*, the parser would get very confused. The parser sees *&name2* and interprets it as a variable that requires immediate substitution. Because it probably will not have a string to substitute in place of *name2*, the parser issues an error and stops processing.

Therefore, we must do something to make the parser pass the variable list through the system intact. To do this, we use the special *&amp* token.

To understand the use of the &amp token, create an example card and process it through the system. For example, this card creates a simple softkey on the screen:

Note In the following example, the semicolon (;) is necessary to separate the &amp sequence from the next token or variable in the string.

```
(Assume that an internal variable exists called Name,
    with the value “George”)

<card id=”MyCard”>
    <keybindings>
        <item id=”&Name”
            onpick=”href=info&amp;amp;event=APP_RELEASE”>Exit
        </item>
    </keybindings>
</card>
```

So, if the application requests *MyCard*, this card comes into the phone and the BTXML parser generates its first approximation of the final card, as shown in the following example:

```
<card id=”MyCard”>
    <keybindings>
        <item id=”George”
            onpick=”href=info&amp;event=APP_RELEASE”>Exit
        </item>
    </keybindings>
</card>
```

In the first pass, the parser replaced the *&Name* variable with the value, *George*. The first *&* is also transformed into the single character *&*. Now the card is passed into the application, where it is parsed again, resulting in the BTXML code shown in the following example:

```
<card id=”MyCard”>
    <keybindings>
        <item id=”George”
            onpick=”href=info&event=APP_RELEASE”>Exit
        </item>
    </keybindings>
</card>
```

In this pass, the remaining *&* is reduced to the single & character expected by the HTML application in the phone.

Tip Any time a card wants to use an & character in a request, it must be replaced with the *&* sequence.

Other Escape Sequences

In addition to the special *&amp* token, there are several other escape sequences that must be used when sending BTXML. See Table 5. These are common among BTXML, CMXML, and standard XML.

Table 5 Common Escape Sequences

Character	Name	Escape Sequence
&	Ampersand	&amp;
“	Quote	&quot;
‘	Apostrophe	&apos;
<	Left bracket	&lt;
>	Right bracket	&rt;

Table 5 Common Escape Sequences (continued)

Character	Name	Escape Sequence
‘ ‘	Nonbreaking space	
char	character	(ascii value for space)
hex char	hex character	(same thing in hexadecimal)

Syntax Hierarchy

This section provides details concerning BXTML attributes, as well as information on which tokens can be used within other tokens.

Table 6 contains all of the BTXML elements and the attributes that they support. It also includes the attribute sizes and default values (where applicable).

Table 6 BTXML Attributes

Element	Attribute	BTXML Size (in bytes unless stated otherwise)
background	href	256; maximum content size of text: 7 lines of 48 characters
br	n/a	
btxml	n/a	
card	id href	32 256 Maximum card size: 8K (if entire deck consists of a single card)
calltimer	callid value	16 12 Maximum calltimers: 6

Table 6 BTXML Attributes (continued)

Element	Attribute	BTXML Size (in bytes unless stated otherwise)
datetime	form: • H:HH—2 digits, 24 hour hours • h:HH—2 digits, 12 hour hours • i:MM—2-digit minutes • s:SS—2-digit seconds • m:MM—2-digit month • d:DD—2-digit day • y:YY—2-digit year • Y:YYYY—4-digit year • M: Name of month: Jan, Feb, and so on • F: Name of month: January, February, and so on • \: Escape character • a: 'a' or 'p', depending on hour • A: 'A' or 'P', depending on hour • :: To add a nonblinking colon, use '\:' in the format string Value	12 (string specifying the date format) UINT_32—date or time (in seconds)
deck	id	32 (maximum deck size: 8K)
font	fore "WHITE|LTGREY|DKGREY|BLACK" (case sensitive) back "WHITE|LTGREY|DKGREY|BLACK" (case sensitive)	Default=BLACK. Invalid values default to BLACK. Default=WHITE. Invalid values default to WHITE.

Table 6 BTXML Attributes (continued)

Element	Attribute	BTXML Size (in bytes unless stated otherwise)
icon	id "WAIT", "RINGOUT", "RING", "TRANSFER", "HOLD", "CONNECTED", "CALLWAIT", "TIME", "BUSY", "CURSOR", "MSGSWAIT", "PROTOCOL", "SETTINGS", "SERVICES", "DIRECTORY", "INFO", "MESSAGES", "MSGREAD", "UNCHECKED", "CHECKED", "LOCKED", "UNLOCKED", "CONTRASTL", "CONTRASTD", "SCRLUP", "SCRLDWN", "SCRLUPDWN", "SCRLOFF", "ONHOOK", "OFFHOOK", "ONURI", "OFFURI", "ONHOOKUN", "OFFHOOKUN", "ONURIUN", "OFFURIUN", "POLYCOM", "RINGER", "HEADSET", "HANDSET", "WORK", "MOBILE", "HOME", "BLOCKED", "NUMPROMPT", "ALPHPROMPT", "RARROW", "CLOCK", "DKSPACE8", "LTSPACE8", "DKSPACE16", "LTSPACE16", "DKSPACE32", "LTSPACE32", "TOPCORNER", "BOTCORNER", "KEYPAD", "KEYPADLOCK" (case sensitive)	
image	height width href {Max images}	UINT_8 (0-250) UINT_8 (0-131) 256 (Total size of all images downloaded must be less than 30K)

Table 6 BTXML Attributes (continued)

Element	Attribute	BTXML Size (in bytes unless stated otherwise)
item	id type "menu|input|hyper|key" (case sensitive)	256
	name value onhilite onpick flags. Valid values are: • A a—Alpha for input items. • B b—Blinking for menu items. • N n—Numeric for input items. • S s—Sticky for keymaps. • D—Dialplan for input items. • - —Passthrough for input items and no autopick for menu items. • *—Password protect for input items. • C c—Place cursor at end of string for input items. • . —Change '*' to '.' for input items. • U u—No number for menu items.	32 256 256 256
	text (content) {Max Linekeys} {Max Softkeys} {Max Hardkeys} {Max MenuItems} {Max InputItems}	256 6 15 28 32 32
keybindings	n/a	
rj	n/a	
set	name value	64 256
slider	id value {Max Sliders}	ignored UINT_8 1
status	n/a	40, including widgets (3 bytes/widget)
timer	time URL {Max Timers}	UINT_16 (Range: 0–6,553) 256 1
title	n/a	64, including widgets (3 bytes/widget)

Table 6 BTXML Attributes (continued)

Element	Attribute	BTXML Size (in bytes unless stated otherwise)
update	{Max UpdateItems}	32
window	width = "wide|narrow" (case sensitive). Wide is used, everything else is interpreted as narrow	
	href	256
	onhilite	256
	{Max Content Size}	Text; 64 lines of 48 characters

Table 7 lists the containment rules for the various BTXML elements. For example, a <background> element can contain a <br> element. A <br> element cannot contain any other element.

Table 7 Containment Rules for Elements

Element	Can Contain
background	br datetime font icon image rj
br	n/a
calltimer	n/a
card	background keybindings status timer topbar window
datetime	n/a
deck	card
font	br datetime icon rj
icon	n/a
image	n/a
item	br calltimer datetime icon image item rj

Table 7 Containment Rules for Elements (continued)

Element	Can Contain
rj	n/a
set	n/a
slider	n/a
status	font icon slider
timer	n/a
title	icon
topbar	n/a
update	item slider
window	br calltimer datetime font icon image item rj slider title

Troubleshooting BTXML Scripts

Errors and mistakes are a normal part of the development process. This section helps you isolate problems in BTXML scripts.

The Command-Line Interpreter

The command-line interpreter (CLI) is available through the Telnet facility of the phone and the serial console port.

The following sections describe the CLI debug commands that are supported in the Cisco 7940/7960 IP telephones.

Note Multiple **debug** commands can be used at the same time.

debug http

The **debug http** command is used to monitor the HTTP transfers into and out of the phone, as in the following example:

```
SIP Phone> debug http
Enabling bug logging on this terminal - use 'tty mon 0' to disable
debugs: http
SIP Phone>
SIP Phone>
SIP Phone> Connect2WWWIPPort called IpAddr[1080450446], port[80],
hostname[12.34.5.67.890]
HTTP RECV (ACK CMD)
HTTP RECV (OPEN CMD)

HTTP Send [147] Bytes of Data
Data Packet is:
===============
GET /btxml/index.pl?prodid=7960&event=APP_INIT HTTP/1.1
User-Agent: Allegro-Software-WebClient/3.10b1
Host: 12.345.67.890
Connection: Close

===============
HTTP RECV (Data Packet)
Http Recv [126] Bytes of Data
Data Packet is:
===============
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 25 Jun 2002 19:18:54 GMT
Connection: close
Content-type: text/xml

===============

HTTP RECV (Data Packet)
Http Recv [956] Bytes of Data
Data Packet is:
===============

    <card>
    <set name="prodid" value="7960"/>
        <window width="wide">
               <item type="menu" onpick="event=APP_REQUEST&amp;a....
               <item type="menu" onpick="event=APP_REQUEST&amp;a....
               <item type="menu" onpick="event=APP_REQUEST&amp;am....
               <item type="menu" onpick="event=APP_REQUEST&amp;a...
           </window>
           <status/>
           <keybindings>
                    <item id="Ask"  onpick="event=KEY_ACCEPT">Select</item>
                    <item id="NULL" onpick="null"></item>
                    <item id="Exit" onpick="event=APP_RELEASE">Exit</item>
           </keybindings>
           </card>

===============

HTTP RECV (CLOSE CMD)
Platform_Close_Socket conn[3]
Platform_Close_Socket conn[3]
```

The example above is the output produced when the services key is pressed. The lines in the downloaded card were truncated to make the output easier to read. The **debug http** command provides a very simple way to see the data being sent back and forth between the phone and server. This can help resolve cases in which a URL is specified incorrectly or in which a problem exists with the server.

debug xml-events

The **debug xml-events** command is used to monitor the events being generated in the phone. Events are used internally to pass information between applications.

This example shows the output of the **debug xml-events** command as a result of pressing the services key:

```
SIP Phone>
SIP Phone> debug xml-events
debugs: xml-events
SIP Phone> XML Event: href=basic, event=SKEY, target=(null), action=services,
card=(null)
XML Event: href=services, event=(null), target=(null), action=(null),
card=services
XML Event: href=(null), event=APP_REQUEST,
target=btxml://12.345.67.890/btxml/index.pl, action=(null), card=(null)
XML Event: href=basic, event=APP_SUSPEND, target=(null), action=(null),
card=(null)
XML Event: href=btxml://12.345.67.890/btxml/index.pl, event=APP_INIT,
target=(null), action=(null), card=(null)
```

The significant events are as follows:

1. A key comes into the basic application (event=SKEY, action=services).
2. The services card is requested (card=services).
3. The index.pl program is started and sent the APP_REQUEST event.
4. The basic application is sent an APP_SUSPEND event.
5. The index.pl program is sent an APP_INIT.

The **debug xml-events** command can be used to determine why a certain action is not taking place as expected. For example, suppose you placed code in your application to react in a certain way when the APP_SUSPEND event is sent. The **debug xml-events** command helps verify that the event is actually sent when expected.

debug xml-deck

The **debug xml-deck** command is used to monitor the decks and cards being referenced by applications. The following example shows the output of the **debug xml-deck** command that results from pressing the services key. It then shows what happens when the exit key is pressed to return to the basic phone application:

```
SIP Phone> debug xml-deck
debugs: xml-deck
SIP Phone> Deck Found: services
Card Found: services
    Deck Found: basic
Card Found: bphome
Deck Found: basic
Card Found: bpbackground
```

This tool is useful for determining whether the cards you are trying to use actually exist and are being accessed properly.

debug xml-vars

The **debug xml-vars** command is used to monitor the variables being used by applications. The following example shows the output of the **debug xml-deck** command as a result of pressing the services key:

```
SIP Phone> debug xml-vars
debugs: xml-vars
SIP Phone> ---LIST---
href=basic
event=SKEY
action=services
------
---LIST---
href=services
card=services
------
---LIST---
event=APP_REQUEST
target=btxml://12.345.67.890/btxml/index.pl
prodid=7960
------
---LIST---
href=basic
event=APP_SUSPEND
------
---LIST---
href=btxml://12.345.67.890/btxml/index.pl
prodid=7960
event=APP_INIT
------
```

debug xml-post

The **debug xml-post** command is used to monitor the text strings sent into the event parser. The following example shows the output of the **debug xml-post** command that results from pressing the services key:

```
SIP Phone> debug xml-post
debugs: xml-post
SIP Phone> XML POST: <event=SKEY&amp;action=services>
VAR: <>=<>
XML POST: <href=services&amp;card=services>
VAR: <(null)>=<(null)>
XML POST:<event=APP_REQUEST&amp;
target=btxml://12.345.67.890/btxml/index.pl&amp;prodid=7960>
VAR: <(null)>=<(null)>
XML POST: <event=APP_SUSPEND>
VAR: <(null)>=<(null)>
XML POST: <prodid=7960&amp;event=APP_INIT>
VAR: <href>=<btxml://12.345.67.890/btxml/index.pl>
```

This command would normally be used in combination with the **debug xml-events** command to ensure that event strings are being parsed as expected.

Parser Errors

In addition to the debug commands shown above, the BTXML parser produces output when syntax errors are found in the cards, as in the following example:

```
SIP Phone> ERROR: Could not find token xyz in current element table
```

Glossary

This section contains a list of terms used in this document. For a more comprehensive list of common terms and acronyms, see *Internetworking Terms and Acronyms*.

Term	Definition
BMP	Bitmap
BTXML	Basic Telephony Extended Markup Language
CMXML	Cisco Call Manager eXtensible Markup Language
GUI	Graphical User Interface
LDAP	Lightweight Directory Access Protocol
MGCP	Media Gateway Control Protocol
SDK	Software Development Kit
UI	User Interface
WAP	Wireless Access Point
XML	eXtensible Markup Language