This appendix contains information on all the frame types that you can create in World of Warcraft, with all the XML attributes and elements as well as all Lua methods and available script handlers. Frame types follow an object-oriented model, meaning that a frame type can inherit attributes and methods from another frame type. You can find an introduction to object-oriented programming in Chapter 6 if you are not familiar with this topic.

You may wonder why there is such an appendix in this book, as you can probably find most of the information covered here on the Internet. WoWWiki has a few good pages that deal with exactly the same topic as this appendix. However, the information there is scattered over multiple pages and many of the articles are outdated or incomplete. This appendix is up to date as of patch 3.1 (the next patches probably won’t change anything important in the frame API) and contains everything you need to know about a particular frame type in one place.

Object

Object is an abstract frame type; that is, you cannot create frames from this type. But this type is the base class for all other frame types as every other type inherits from Object. This means that all of the methods, XML attributes, and XML elements listed here are available for all frames.

This frame type is sometimes also called UIObject, which is probably the type’s internal name used in the C++ code of the game.

XML Attributes

The Object type has the following XML attributes.

alpha

Defines the frame’s transparency (0 – 1). Default: 1.
parent
Defines the frame's parent.

name
Defines the frame's name.

inherits
The template to use.

virtual
Creates a template. Default: false.

Lua Methods
The Object type has the following Lua methods.

object:GetAlpha()
Returns the frame’s alpha value.

object:GetEffectiveAlpha()
Returns the frame's effective alpha value by multiplying its alpha value with the parent’s effective alpha.

object:GetName()
Returns the name of the frame.

object:GetObjectType()
Returns the type of the object.

object:GetParent()
Returns the parent of the frame.

object:IsObjectType(objType)
Returns 1 if object is of the type objType, or of a type that inherits from objType. For example, CreateFrame("CheckButton"):IsObjectType("Button") returns 1 because the type CheckButton inherits from the type Button.

object:SetAlpha()
Sets the frame’s transparency (0 – 1).
object: SetParent(obj)
Sets the frame’s parent.

Font
Font is an object that inherits from Object and represents a font that can be used by FontStrings or Buttons.

XML Attributes
The Font type has the following XML attributes.

font
The path to the font file you want to use.

justifyH
Sets the horizontal justification setting. This can either be LEFT, RIGHT, or CENTER. Default: CENTER.

justifyV
Sets the vertical justification setting. This can either be BOTTOM, TOP, or MIDDLE. Default: MIDDLE.

monochrome
Uses a monochrome font. Default: false.

outline
The outline to use; can be NONE, NORMAL, or THICK. Default: NONE.

spacing
The spacing between characters. Default: 0.

XML Elements
The Font type has the following XML elements.

FontHeight
Controls the height of the font with its Value attribute.

Color
Controls the color of the font with its r, g, b, and a attributes.
Shadow
Contains the two elements Color and Offset. Color controls the color with the attributes r, g, b, and a, and Offset controls the shadow offset with one of the elements AbsDimension or RelDimension.

Lua Methods
The Font type has the following Lua methods.

```lua
font:CopyFontObject(otherObject)
```
Copies all settings from `otherObject`.

```lua
font:GetFont()
```
Returns the font file used.

```lua
font:GetFontObject()
```
Returns the font object itself. This method makes sense when called from a frame that inherits from Font.

```lua
font:GetJustifyH()
```
Returns the horizontal alignment of the text.

```lua
font:GetJustifyV()
```
Returns the vertical alignment of the text.

```lua
font:GetMultilineIndent()
```
Returns the multiline indentation setting.

```lua
font:GetShadowColor()
```
Returns the color of the font’s shadow.

```lua
font:GetShadowOffset()
```
Returns the offset of the shadow.

```lua
font:GetSpacing()
```
Returns the spacing between characters.

```lua
font:GetTextColor()
```
Returns the color of the text.
**font::SetFont(path)**
Sets the font file to use.

**font::SetFontObject(font)**
This method makes sense only when called from a frame (such as a MessageFrame) that inherits from a Font. This method can then set or change the used font object for the text of the frame.

**font::SetJustifyH(alig)**
Sets the horizontal justification setting. The *align* argument can be LEFT, RIGHT, or CENTER.

**font::SetJustifyV(alig)**
Sets the vertical justification setting. The *align* argument can be TOP, BOTTOM, or MIDDLE.

**font::SetMultilineIndent(indent)**
Controls whether to indent the continuation of a string that does not fit in a single line.

**font::SetShadowColor(r, g, b, a)**
Sets the color of the shadow.

**font::SetShadowOffset(x, y)**
Sets the offset of the shadow.

**font::SetSpacing(n)**
Sets the spacing between characters.

**font::SetTextColor(r, g, b, a)**
Sets the color of the text.

## Region
Region inherits from Object. The only way to create a Region is by calling frame::CreateTitleRegion() or using the XML element TitleRegion of a frame; you cannot use CreateFrame to create a Region. It is the base class for everything that is visible on your screen (Frame, Texture, and so on), as it defines sizes and anchors.

## XML Attributes
The Region type has the following XML attributes.

**setAllPoints**
Fits the frame into its parent. Default: false.
hidden
Defines whether the frame starts visible or hidden. Default: false.

**XML Elements**
The Region type has the following XML elements.

**Size**
Size consists of either the element AbsDimension or RelDimension. The former refers to an absolute size that depends only on the effective scale of the frame, while the latter refers to a size relative to the whole screen. Both have the attributes \( x \) and \( y \) that specify the actual size. A detailed description can be found in Chapter 5.

**Anchors**
Consists of one or more Anchor elements. An anchor has the attributes Point, RelativePoint, and RelativeTo and the element Offset which, like Size, consists of AbsDimension or RelDimension and specifies the offset. A detailed description can be found in Chapter 5.

**Lua Methods**
The Region type has the following Lua methods.

region:ClearAllPoints()
Removes all points but does not change the position. You have to set a new point in order to move the frame.

region:getBottom()
Returns the \( y \) coordinate of the frame’s bottom.

region:getCenter()
Returns the \( x \) and \( y \) coordinates of the frame’s center.

region:getHeight()
Retrieves the frame’s height.

region:getLeft()
Returns the \( x \) coordinate of the frame’s left border.

region:getNumPoints()
Returns the number of anchor points.
region:GetPoint(n)
Retrieves the nth point of the frame. The return values are equivalent to the arguments passed to SetPoint when creating the anchor.

region:GetRect()
Returns the frame’s bottom left x and y coordinates followed by the width and height.

region:GetTop()
Returns the y coordinate of the frame’s top.

region:GetWidth()
Returns the width of the frame.

region:Hide()
Hides the frame.

region:IsProtected()
Returns 1 if the frame is a protected frame.

region:IsShown()
Determines whether the frame is shown. This does not take the frame’s parent into account; see IsVisible.

region:IsVisible()
Determines whether the frame and all of its parents (and grandparents, and so on) are shown.

region:SetAllPoints(frame)
Stretches the frame to fit into frame.

region:SetHeight(height)
Sets the frame’s height.

region:SetPoint(point, relativeFrame, relativePoint, x, y)
Adds an anchor point. See Chapter 5 for a detailed description.

region:SetWidth(width)
Sets the frame’s width.
region:StopAnimating()
Stops all currently playing animations.

region:GetAnimationGroups()
Returns all animation groups of the frame.

region:CreateAnimationGroup(name, inherits)
Creates a new animation group for the frame.

region:IsDragging()
Determines whether the user is currently dragging the frame or its parent.

region:CanChangeProtectedState()
Determines whether you are allowed to call a protected method from tainted code at the moment.

**Frame**
Frame inherits from Region, which means it has all the methods and attributes of Object and Region.

**XML Attributes**
The Frame type has the following XML attributes.

toplevel
Fits the frame into its parent. Default: false.

movable
Defines whether the frame starts visible or hidden. Default: false.

resizable
Makes the frame resizable. Default: false.

frameStrata
Defines the frame strata. Frame strata are described in Chapter 5.

framelevel
Defines the level of the frame.
enableMouse
Enables mouse input. Default: false.

enableKeyboard
Enables keyboard input. Default: false.

clampedToScreen
Prevents the frame from being dragged off the screen. Default: false.

jumpNavigateEnabled
Allows the element to be selected with a gamepad or joystick.

jumpNavigateStart
Sets the frame as first frame to be selected when using a gamepad or joystick. Default: false.

XML Elements
The Frame type has the following XML elements.

TitleRegion
A frame of the type Region, which is used as the title region of the frame.

ResizeBounds
Consists of the two elements MinResize and MaxResize, which can contain an AbsDimension or a RelDimension element defining the minimum and maximum size of the frame.

Backdrop
Defines the backdrop. A detailed description is available in Chapter 5.

HitRectInsets
Sets the rectangle in which the frame accepts mouse clicks via the attributes left, right, top, and bottom.

Layers
Contains multiple Layer elements, which in turn may contain multiple Textures or FontStrings. A detailed description is available in Chapter 5.

Frames
Contains multiple frames of any type that inherits from Frame.
Scripts
Contains the script handlers.

Lua Methods
The Frame type has the following Lua methods.

`frame:AllowAttributeChanges()`
A protected method. Allows tainted code to change attributes in the current execution path while in combat.

`frame:CreateFontString(name, layer, inheritsFrom)`
Creates a new FontString object in the layer `layer` as child of the frame.

`frame:CreateTexture()`
Creates a new Texture object in the layer as child of the frame.

`frame:CreateTitleRegion()`
Creates a new TitleRegion, a frame of the type Region. A title region comes with predefined script handlers (unchangeable, because SetScript doesn’t exist for Regions) for OnDragStart and OnDragStop; these handlers call StartMoving and StopMovingOrSizing respectively on the Region’s parent if it is movable.

`frame:DisableDrawLayer(layerLevel)`
Hides the given layer level.

`frame:EnableDrawLayer(layerLevel)`
Displays the given layer level.

`frame:EnableJoystick()`
Enables joystick and gamepad input for the frame.

`frame:EnableKeyboard()`
Enables keyboard events.

`frame:EnableMouse()`
Enables mouse events.

`frame:EnableMouseWheel()`
Enables mouse wheel events.
frame:GetAttribute(attr)
Returns the value of attr.

frame:GetBackdrop()
Returns the frame’s backdrop as a table in the same format as SetBackdrop expects it.

frame:GetBackdropBorderColor()
Returns the color of the backdrop’s border.

frame:GetBackdropColor()
Returns the color of the backdrop.

frame:GetBoundsRect()
Like GetRect but takes the frame’s layers into account.

frame:GetChildren()
Returns how many children the frame has.

frame:GetClipRectInsets()
Returns all children of the frame.

frame:GetDepth()
Get the frame’s depth on 3D monitors.

frame:GetEffectiveDepth()
Returns the frame’s effective depth by taking the parent’s depth into account.

frame:GetEffectiveScale()
Returns the effective scale of the frame.

frame:GetFrameLevel()
Returns the frame level, which determines whether a frame is in front of or behind another frame in the same stratum.

frame:GetFrameStrata()
Returns the stratum the frame is in.

frame:GetHitRectInsets()
Returns the rectangle that defines where the frame can be clicked.
frame:GetMaxResize()
Returns the maximum width and height the frame can be resized to by the user.

frame:GetMinResize()
Returns the minimum width and height.

frame:GetNumChildren()
Returns all children of the frame.

frame:GetNumRegions()
Returns the number of regions in the frame.

frame:GetRegions()
Returns all regions (Textures and FontStrings) of the frame.

frame:GetScale()
Returns the frame’s scale.

frame:GetScript(script)
Returns the function assigned to ScriptHandler.

frame:GetTitleRegion()
Returns the title region of the frame.

frame:HasScript(script)
Determines whether the frame can have the script handler script.

frame:HookScript(script, func)
Hooks the script handler script with the function func.

frame:IgnoreDepth(ignore)
Displays the frame in the default depth if ignore is true.

frame:IsClampedToScreen()
Determines whether the frame is clamped to the screen.

frame:IsEventRegistered(event)
Determines whether event is registered.
frame:isIgnoringDepth()
Returns true if IgnoreDepth was called with true.

frame:isJoystickEnabled()
Returns true if the frame accepts joystick and gamepad input.

frame:isJumpNavigateEnabled()
Determines whether the frame can be selected with a joystick or gamepad.

frame:isJumpNavigateStart()
Returns 1 if the frame is the first frame that is selected in an environment with a gamepad or joystick instead of a mouse.

frame:isKeyboardEnabled()
Returns 1 if the frame receives keyboard events.

frame:isMouseEnabled()
Returns 1 if the frame receives mouse events.

frame:isMouseWheelEnabled()
Returns 1 if the frame receives mouse wheel events.

frame:isMovable()
Returns 1 if the frame is movable.

frame:isResizable()
Returns 1 if the frame is resizable.

frame:isTopLevel()
Returns 1 if the frame is a top-level frame.

frame:isUserPlaced()
Returns 1 if the userplaced attribute is set.

frame:lower()
Reduces the frame’s level.

frame:raise()
Increases the frame’s level.
frame:RegisterAllEvents()
Registers all available events for debugging purposes.

frame:RegisterEvent(event)
Registers the given event.

frame:RegisterForDrag(btn1, ...)
Enables the OnDragStart/Stop handlers for the given mouse buttons.

frame:SetAttribute(attribute, value)
Sets attribute to value.

frame:SetBackdrop(backDrop)
Sets the frame’s backdrop; backDrop is a table in the following format:

```
{  
  bgFile = "bgFile", edgeFile = "edgeFile", tile = false,  
  tileSize = 16, edgeSize = 32,  
  insets = {left = 0, right = 0, top = 0, bottom = 0}  
}
```

A description of backdrops can be found in Chapter 5.

frame:SetBackdropBorderColor(r, g, b, a)
Sets the border color of the backdrop.

frame:SetBackdropColor(r, g, b, a)
Sets the background color of the background.

frame:SetClampRectInsets(x, y, width, height)
Sets the rectangle that is used to determine whether the frame is on or off the screen.

frame:SetClampedToScreen(clamped)
Prevents the user from dragging the frame off the screen.

frame:SetDepth(depth)
Sets the frame’s depth on 3D monitors.

frame:SetFrameLevel(level)
Sets the frame’s level.
frame: setFrameStrata(strata)
Sets the frame’s strata.

frame: setHitRectInsets(x, y, width, height)
Sets the rectangle in which the frame accepts mouse clicks.

frame: setID(id)
Sets the ID of the frame.

frame: setJumpNavigateEnabled(enabled)
Makes the frame selectable with a gamepad or joystick.

frame: setMaxResize(width, height)
Sets the maximum size of the frame.

frame: setMinResize(width, height)
Sets the minimum size of the frame.

frame: setMovable(movable)
Makes the frame movable with StartMoving.

frame: setResizable(movable)
Makes the frame resizable.

frame: setScale(scale)
Sets the scale of the frame.

frame: setScript(script, func)
Sets the script handler for script to func.

frame: setTopLevel(topLevel)
Makes the frame a top-level frame.

frame: setUserPlaced(userPlaced)
Saves the position of the frame.

frame: StartMoving()
Attaches the frame to your mouse cursor.
frame:StartSizing()
Starts to resize the frame.

frame:StopMovingOrSizing()
Stops resizing or moving the frame.

frame:UnregisterAllEvents()
Unregisters all events.

frame:UnregisterEvent(event)
Unregisters the given event.

Script Handlers
The Frame type has the following script handlers.

   OnAnimFinished(self)
   OnAttributeChanged(self, attributeName, value)
   OnCursorChanged(self, x, y, w, h)
   OnDoubleClick(self, button)
   OnDragStart(self, button)
   OnDragStop(self)
   OnEnter(self, motion)
   OnEnterPressed(self)
   OnEscapePressed(self)
   OnEvent(self, event, ...)
   onHide(self)
   OnKeyDown(self, key)
   OnKeyUp(self, key)
   OnLeave(self, motion)
   OnLoad(self)
   OnMouseDown(self, button)
   OnMouseUp(self, button)
   OnMouseWheel(self, delta)
   OnReceiveDrag(self)
   OnShow(self)
   OnSizeChanged(self, width, height)
FontString

FontString inherits from Region. You cannot create a FontString via CreateFrame; you have to create it by calling the method CreateFontString of a frame object or by defining it in the <Layers> element in the XML description of a frame.

FontStrings have all methods of Fonts except for CopyFontObject. However, FontStrings do not inherit from Font; fontstring:IsObjectType("Font") returns nil.

XML Attribute

The FontString type has only one XML attribute.

text
The text on the button.

Lua Methods

The FontString type has the following Lua methods.

fontString:CanNonSpaceWrap()
Returns 1 if the font string is set to wrap in the middle of a word.

fontString:GetDrawLayer()
Returns the layer the font string is in. Layers are explained in detail in Chapter 5.

fontString:GetStringHeight()
Returns the height needed by the font string.

fontString:GetStringWidth()
Returns the width of the font string.

fontString:GetText()
Returns the displayed text.

fontString:SetAlphaGradient(start, length)
Creates an alpha gradient starting at start over length pixels.
**fontString::SetDrawLayer(layer)**
Changes the draw layer.

**fontString::SetFormattedText(fmtStr, ...)**
Sets formatted text.

**fontString::SetNonSpaceWrap(nonSpace)**
Controls whether the font string is allowed to wrap in the middle of a string.

**fontString::SetText(text)**
Sets the text.

**fontString::SetTextHeight(height)**
Sets the text height.

**fontString::SetVertexColor(r, g, b, a)**
Sets the vertex color of the font string.

**Texture**
Texture inherits from Region. You cannot create a Texture via CreateFrame; you have to create it by calling the method CreateTexture of a frame object or by defining it in the `<Layers>` element in the XML description of a frame.

**XML Attributes**
The Texture type has the following XML attributes.

**alphaMode**
Controls the effect of the texture’s alpha channel. This can be one of the following values: DISABLE, BLEND (default), ALPHAKEY, ADD, or MOD.

**file**
The file containing the texture to display.

**XML Elements**
The Texture type has the following XML elements.

**Color**
Sets the texture to a solid color, with the attributes r, g, b, and a.
Gradient

Creates a gradient on the texture; the attribute orientation of this element controls the orientation and can either be HORIZONTAL (default) or VERTICAL. The two elements MinColor and MaxColor of the Gradient element set the start and end color.

TexCoords

Controls the texture coordinates with the attributes left, right, top, and bottom.

Lua Methods

The Texture type has the following Lua methods.

```
texture:GetBlendMode()
Returns the blend mode.

texture:GetDrawLayer()
Returns the draw layer.

texture:GetTexCoordModifiesRect()
Returns 1 if texture coordinates modify the display rectangle and nil if they stretch the texture.

texture:GetTexture()
Returns the path of the texture used.

texture:GetVertexColor()
Returns the vertex color.

texture:IsDesaturated()
Returns 1 if the texture is desaturated.

texture:SetBlendMode()
Sets the blend mode.

texture:SetDesaturated(desaturated)
Changes the desaturation state of the texture.

texture:SetDrawLayer(layer)
Changes the draw layer.
texture:SetGradient(orientation, startR, startG, startB, endR, endG, endB)
Creates a gradient over the whole texture. The orientation argument can be either HORIZONTAL or VERTICAL.

texture:SetGradientAlpha(orientation, startR, startG, startB, startA, endR, endG, endB, endA)
Same as SetGradient but with an additional alpha value.

texture:SetRotation(angle, cx, cy)
Rotates the texture angle degrees around the center point cx/cy, which is relative to the center of the texture. Default values for cx and cy are 0.

texture:SetTexCoord(left, right, bottom, top)
Sets texture coordinates. Chapter 8 contains a detailed description.

texture:SetTexCoordModifiesRect(modifies)
Sets whether texture coordinates change the displayed part of the texture or stretch the texture.

texture:SetTexture(file or r, g, b, a)
Loads and displays the provided files or fills the texture with a solid color.

texture:SetVertexColor(r, g, b, a)
Sets the vertex color of the texture. You can think of the vertex color as a colored light shining on the texture.

**Button**

Button inherits from Frame; it can be used to create clickable buttons.

**XML Attributes**

The Button type has only one XML attribute.

**text**

The text on the button.

**XML Elements**

The Button type has the following XML elements.
ButtonText
The font string that is used for the button’s text.

DisabledColor
A color object (uses r, g, b, and a attributes); sets the vertex color for the disabled texture.

DisabledFont
Sets the font for the disabled button in its style attribute (expects the name of a Font object).

DisabledTexture
A texture object; sets the disabled button texture.

HighlightColor
A color object; sets the vertex color for the highlight texture.

HighlightFont
Sets the font for the highlighted button in its style attribute (expects the name of a Font object).

HighlightTexture
A texture object. The highlight button texture.

NormalFont
Sets the font for the normal button in its style attribute (expects the name of a Font object).

NormalTexture
A texture object; sets the normal button texture.

PushedTextOffset
A dimension object that defines the text offset when the button is pushed.

PushedTexture
A texture object; sets the pushed button texture.

Lua Methods
The Button type has the following Lua methods.

button:Click()
Executes the button’s OnClick handler.
button:Disable()
Disables the button.

button:Enable()
Enables the button.

button:GetButtonState()
"PUSHED" or "NORMAL".

button:GetDisabledFontObject()
Returns the disabled font object.

button:GetDisabledTexture()
Returns the disabled texture.

button:GetFontString()
Returns the used font string.

button:GetHighlightFontObject()
Returns the highlight font object.

button:GetHighlightTexture()
Returns the highlight texture.

button:GetNormalFontObject()
Returns the normal font object.

button:GetNormalTexture()
Returns the normal texture.

button:GetPushedTextOffset()
Returns the x and y offsets of the pushed text.

button:GetPushedTexture()
Returns the pushed texture.

button:GetText()
Returns the displayed text.
button: GetTextHeight()
Returns the text's height.

button: GetTextWidth()
Returns the text's width.

button: IsEnabled()
Checks whether the button is enabled.

button: LockHighlight()
Shows the highlight texture.

button: RegisterForClicks(btn1, ...)
Registers the given mouse buttons for click events.

button: SetButtonType()
Sets the button's state ("NORMAL" or "PUSHED").

button: SetDisabledFontObject(font)
Sets the disabled font object.

button: SetDisabledTexture(texture)
Sets the disabled texture.

button: SetFontString(fontStr)
Sets the FontString object used.

button: SetFormattedText(formatStr, ...)
Sets formatted text.

button: SetHighlightFontObject(font)
Sets the highlight font object.

button: SetHighlightTexture(texture)
Sets the highlight texture.

button: SetNormalFontObject(font)
Sets the normal font object.
button: SetNormalTexture(texture)
Sets the normal texture.

button: SetPushedTextOffset(x, y)
Sets the pushed text’s offsets.

button: SetPushedTexture(texture)
Sets the pushed texture.

button: SetText(text)
Sets the displayed text.

button: UnlockHighlight()
Hides the highlight texture.

Script Handlers
The Button type has the following script handlers.

    PostClick(self, button)
    PreClick(self, button)
    OnClick(self, button, down)

CheckButton
CheckButton inherits from Button and represents a check box.

XML Attributes
The CheckButton type has only one XML attribute.

    checked
Can be set to true to check the button by default.

XML Elements
The CheckButton type has the following XML elements.

    DisabledCheckedTexture
A texture object; sets the texture that is shown on disabled checked buttons.
CheckedTexture
A texture object; sets the texture that is shown on normal checked buttons.

Lua Methods
The CheckButton type has the following Lua methods.

`checkButton:GetChecked()`
Returns 1 if the button is checked.

`checkButton:GetCheckedTexture()`
Returns the texture object of checked buttons.

`checkButton:GetDisabledCheckedTexture()`
Returns the texture object of disabled checked buttons.

`checkButton:SetChecked(checked)`
Sets the check button to checked.

`checkButton:SetCheckedTexture(texture)`
Sets the normal checked texture.

`checkButton:SetDisabledCheckedTexture(texture)`
Sets the disabled checked texture.

Cooldown
Cooldown is the cooldown display that can be seen on action bars. It inherits from Frame.

XML Attributes
The Cooldown type has the following XML attributes.

`reverse`
Set this to true for a reverse cooldown. Default: false.

`drawEdge`
Set this to true to draw edges around the cooldown display. Default: false.

Lua Methods
The Cooldown type has the following Lua methods.
cooldown:SetCooldown(start, duration)
Displays a cooldown that started at start (timestamp as returned by GetTime() and GetSpellCooldown(spellName)) with the given duration in seconds.

cooldown:SetReverse(reverse)
Set this to true for a reverse cooldown.

**ColorSelect**
ColorSelect is a frame that allows the user to pick a color. This frame type inherits from Frame.

**XML Elements**
The ColorSelect type has the following XML attributes.

*ColorWheelTexture*
A texture object that is used for the color wheel.

*ColorWheelThumbTexture*
A texture object that is used for the color wheel thumbnail.

*ColorValueTexture*
A texture object that is used for the color selector.

*ColorValueThumbTexture*
A texture object that is used for the color selector thumbnail.

**Lua Methods**
The ColorSelect type has the following Lua methods.

`colorSelect:GetColorHSV()`
Returns the selected color as HSV (hue, saturation, value) values.

`colorSelect:GetColorRGB()`
Returns the selected color as RGB.

`colorSelect:GetColorValueTexture()`
Returns the color value texture object.
colorSelect: getColorValueThumbTexture()
Returns the color value thumbnail texture object.

colorSelect: getColorWheelTexture()
Returns the color wheel texture.

colorSelect: getColorWheelThumbTexture()
Returns the color wheel thumbnail texture.

colorSelect: setColorHSV(h, s, v)
Sets the selected color to the given HSV value.

colorSelect: setColorRGB(r, g, b)
Sets the selected color to the given RGB value.

Script Handlers
The ColorSelect type has only one script handler.

OnColorSelect(self, r, g, b)

EditBox
EditBox is a frame that allows the user to enter text. It inherits from Frame. EditBoxes have all methods of Fonts except for CopyFontObject. However, EditBoxes do not inherit from Font; editBox: IsObjectType("Font") returns nil.
XML Attributes
The `EditBox` type has the following XML attributes.

`autofocus`
Automatically grabs the focus when it is shown. Default: false.

`blinkSpeed`
The blink speed of the cursor. Default: 0.5.

`historyLines`
Determines how many history lines will be saved. Default: 0.

`ignoreArrows`
Ignores the arrow keys so you can still move with them while the edit box has the focus. Default: false.

`letters`
The maximum number of allowed letters. Default 0: (meaning no limit).

`multiline`
Creates a multiple-line edit box. Default: false.

`numeric`
Accepts only numbers. Default: false.

XML Elements
The `EditBox` type has the following two XML attributes.

`HighlightColor`
A color element that sets the color of highlighted text with its `r`, `g`, and `b` attributes.

`TextInsets`
Controls the padding between the text and border with its `left`, `right`, `top`, and `bottom` attributes.

Lua Methods
The `EditBox` type has the following Lua methods.

`editBox:AddHistoryLine(text)`
Adds `text` to the history.
editBox: ClearFocus()
Removes the focus from the edit box.

editBox: GetAltArrowKeyMode()
Returns 1 if the edit box ignores the arrow keys.

editBox: GetBlinkSpeed()
Returns the blink speed.

editBox: GetCursorPosition()
Returns the cursor position.

editBox: GetHistoryLines()
Returns the maximum number of history lines.

editBox: GetIndentedWordWrap()
Returns the indented word wrap setting.

editBox: GetInputLanguage()
Returns the input language; "ROMAN" for non-Asian clients.

editBox: GetMaxBytes()
Returns the maximum number of bytes the edit box can store.

editBox: GetMaxLetters()
Returns the maximum number of letters the edit box can store.

editBox: GetNumLetters()
Returns the current number of letters.

editBox: GetNumber()
Tries to interpret the entered text as a number.

editBox: GetTextInsets()
Returns the text insets.

editBox: HasFocus()
Returns 1 if the edit box currently has the focus.
editBox:HighlightText(start, end)
Highlights the text from start to end. Omit the arguments to highlight the whole text.

editBox:Insert(text)
Inserts text at the current position.

editBox:IsAutoFocus()
Returns 1 if the edit box automatically grabs the focus.

editBox:IsIMECompositionMode()
Returns 1 if the IME mode is enabled. This is a feature for Asian clients.

editBox:IsMultiLine()
Returns 1 for multiple-line edit boxes.

editBox:IsNumeric()
Returns 1 for numeric edit boxes.

editBox:IsPassword()
Returns 1 for password edit boxes.

editBox:SetAltArrowKeyMode(ignoreArrows)
Sets the option whether to ignore arrow keys.

editBox:SetAutoFocus(autoFocus)
Sets the auto focus option.

editBox:SetBlinkSpeed(blinkSpeed)
Sets the cursor blink speed.

editBox:SetCursorPosition(pos)
Sets the cursor position.

editBox:SetFocus()
Grabs the focus.

editBox:SetHistoryLines(n)
Sets the maximum number of history lines.
**editBox: SetIndentedWordWrap**(*wrap*)
Sets the word wrap setting.

**editBox: SetMaxBytes**(*n*)
Sets the maximum number of bytes accepted by the edit box.

**editBox: SetMaxLetters**(*n*)
Sets the maximum number of letters accepted by the edit box.

**editBox: SetMultiline**(*multiline*)
Creates a multiple-line edit box.

**editBox: SetNumber**(*n*)
Sets a numeric edit box to *n*.

**editBox: SetNumeric**()
Makes the edit box accept only numbers.

**editBox: SetPassword**()
Hides entered characters.

**editBox: SetTextInsets**(*left, right, top, bottom*)
Sets the padding between the text and the border of the edit box.

**editBox: ToggleInputLanguage**(*language*)
Uses another input language; only important for Asian clients.

**Script Handlers**
The EditBox type has the following script handlers.

- `OnChar(self, char)`
- `OnCharComposition(self, char)`
- `OnCursorPositionChanged(self, x, y, width, height)`
- `OnEditFocusGained(self)`
- `OnEditFocusLost(self)`
- `OnInputChangeLanguageChanged(self, language)`
- `OnTextChanged(self)`
- `OnTextSet(self)`
**GameTooltip**

GameTooltip frames can be used as tooltips. This type inherits from Frame. Tooltips are automatically resized as you add more lines with their methods.

**Lua Methods**

The GameTooltip method has the following Lua methods.

- **gameTooltip:AddDoubleLine(left, right, lR, lG, lB, rR, rG, rB)**
  Adds a line with two font strings to the tooltip. lR, lG, and lB control the color of the left text left and rR, rG, and rB the color of the right text code inline.

- **gameTooltip:AddFontStrings(left, right)**
  Adds a double line consisting of two font string objects to the tooltip.

- **gameTooltip:AddLine(text, r, g, b)**
  Adds a new single line.

- **gameTooltip:AddTexture(texturePath)**
  Adds the given texture file.

- **gameTooltip:AppendText(text)**
  Appends text to the end of the last line.

- **gameTooltip:FadeOut()**
  Fades the tooltip and hides it.

- **gameTooltip:ClearLines()**
  Removes all content from the tooltip.

- **gameTooltip:GetAnchorType()**
  Returns the current anchor type. These types are explained in Chapter 4.

- **gameTooltip:GetItem()**
  Returns the name and link of the currently displayed item.

- **gameTooltip:GetMinimumWidth()**
  Returns the minimum width of the tooltip.
gameTooltip:GetOwner()
Returns the tooltip’s owner.

gameTooltip:GetPadding()
Returns the padding to other objects.

gameTooltip:GetSpell()
Returns the currently displayed spell.

gameTooltip:GetUnit()
Returns the currently displayed unit.

gameTooltip:IsEquippedItem()
Returns 1 if the tooltip displays a currently equipped item.

gameTooltip:IsOwned()
Returns 1 if the tooltip has an owner.

gameTooltip:IsUnit()
Returns 1 if the tooltip displays a unit.

gameTooltip:NumLines()
Returns the maximum number of lines.

gameTooltip:SetAction(slotId)
Shows the tooltip of a given action button slot.

gameTooltip:SetAnchorType(anchortype, xoffset, yoffset)
Sets an anchor type.

**Setting Tooltip Contents**
The following methods deal with setting the tooltip to specific items or spells. There is no
description of these methods, as the names and arguments are self-explanatory and the func-
tions are rarely needed.

gameTooltip:SetAuctionItem(index)
gameTooltip:SetAuctionSellItem()
gameTooltip:SetBackpackToken(id)
gameTooltip:SetBagItem(bad, slot)
gameTooltip:SetBuybackItem(id)
gameTooltip:SetCurrencyToken(id)
gameTooltip:SetEquipmentSet(id)
gameTooltip:SetExistingSocketGem(id)
gameTooltip:SetGlyph(id)
gameTooltip:SetGuildBankItem(id)
gameTooltip:SetHyperlink(link)
gameTooltip:SetHyperlinkCompareItem(link)
gameTooltip:SetInboxItem(id)
gameTooltip:SetInventoryItem(id)
gameTooltip:SetLootItem(id)
gameTooltip:SetLootRollItem(id)
gameTooltip:SetMerchantCostItem(id)
gameTooltip:SetMerchantItem(id)
gameTooltip:SetPetAction(id)
gameTooltip:SetQuestItem(quest, id)
gameTooltip:SetQuestLogItem(id)
gameTooltip:SetQuestLogRewardSpell(id)
gameTooltip:SetQuestLogSpecialItem(id)
gameTooltip:SetQuestRewardSpell(id)
gameTooltip:SetSendMailItem(id)
gameTooltip:SetShapeshift(id)
gameTooltip:SetSocketGem(id)
gameTooltip:SetSocketedItem(id)
gameTooltip:SetSpell(spellId)
gameTooltip:SetTalent(id)
gameTooltip:SetTotem(id)
gameTooltip:SetTracking(id)
gameTooltip:SetTradePlayerItem(id)
gameTooltip:SetTradeSkillItem(id)
gameTooltip:SetTradeTargetItem(id)
gameTooltip:SetTrainerService(id)
gameTooltip:SetUnit(uId)
gameTooltip:SetUnitAura(uId, i, filter)
gameTooltip:SetUnitBuff(uId, i, filter)
gameTooltip:SetUnitDebuff(uId, i, filter)

gameTooltip:SetMinimumWidth(width)
Sets the minimum width.

gameTooltip:SetOwner(frame)
Sets the owner of the tooltip. A tooltip is hidden when its owner is hidden.

gameTooltip:SetPadding()
Sets the padding.

**MessageFrame**

MessageFrame is a frame that displays a few font strings; older font strings are hidden as you append new strings. It looks like a chat frame but without the scroll bars.

MessageFrame has all methods of Fonts except for CopyFontObject. However, MessageFrame does not inherit from Font; messageFrame:IsObjectType("Font") returns nil.

**XML Attributes**
The MessageFrame method has the following XML attributes.

displayDuration
The time after which an element is removed.

fade
Enables the fade out effect. Default: true.

fadeDuration
Duration of the fade effect. Default: 3.0.

font
The font used.

insertMode
Where to insert new messages; TOP or BOTTOM. Default: BOTTOM.
XML Elements

The MessageFrame type has one available XML element.

textInsets
Controls the padding of the displayed text with the attributes \textit{left}, \textit{right}, \textit{top}, and \textit{bottom}.

Lua Methods

The MessageFrame type has the following Lua methods.

\texttt{messageFrame:AddMessage(msg, r, g, b)}
Adds a message in the given color.

\texttt{messageFrame:Clear()}
Clears the frame.

\texttt{messageFrame:GetFadeDuration()}
Returns the fade duration.

\texttt{messageFrame:GetFading()}
Returns the duration of the fade effect.

\texttt{messageFrame:GetInsertMode()}
Returns the insert mode.

\texttt{messageFrame:GetTimeVisible()}
Returns the time after which a message is hidden.

\texttt{messageFrame:SetFadeDuration(fadeDuration)}
Sets the duration of the fade effect.

\texttt{messageFrame:SetFading(fadeOut)}
Enables or disables the fade effect.

\texttt{messageFrame:SetInsertMode(insertMode)}
Sets the insert mode (TOP or BOTTOM).

\texttt{messageFrame:SetTimeVisible(t)}
Sets the time after which a message is hidden.
Model
Model is a frame that inherits from Frame and displays a 3D model.

XML Attributes
The Model type has the following XML attributes.

file
The path of the model to display.

fogNear
The near fog distance. Default: 0.0.

fogFar
The far fog distance. Default: 1.0.

glow
The glow intensity. Default: 1.0.

scale
The scale of the model. This is a different value than the SetScale/GetScale scale.

XML Elements
The Model type has only one XML attribute.

FogColor
Controls the color of the fog with its r, g, b, and a attributes.

Lua Methods
The Model type has the following Lua methods.

model:AdvanceTime()
This must be called by the OnUpdate handler if you want to play animations in the model.

model:ClearFog()
Removes the fog.

model:ClearModel()
Removes the model.
model:GetFacing()
Returns the direction the model is looking in.

model:GetFogColor()
Returns the color of the fog.

model:GetFogFar()
Returns the maximum fog distance.

model:GetFogNear()
Returns the minimum fog distance.

model:GetLight()
Returns the current light situation. The return values are the same as the arguments to SetLight.

model:GetModel()
Returns the model used.

model:GetModelScale()
Returns the scale of the model.

model:GetPosition()
Returns the position of the model inside the displayed scene.

model:ReplaceIconTexture()
Unknown.

model:SetCamera(id)
Sets the camera to a camera that is defined in the model file.

model:SetFacing(facing)
Sets the facing to a radian value.

model:SetFogColor(r, g, b, a)
Sets the color of the fog.

model:SetFogFar()
Sets the fog’s far clip distance.
PlayerModel

PlayerModel is derived from Model and can be used to display players and NPCs.

Lua Methods

The PlayerModel type has the following Lua methods.

playerModel:RefreshUnit()

Refreshes the currently displayed unit; for example, to update the worn equipment.

playerModel:SetCreature(creatureId)

Sets the model to display the given creature. This only works on creatures in your local cache, meaning creatures you have already seen.
playerModel:SetRotation(rotation)
Rotates the model.

playerModel:SetUnit(unitID)
Displays the model of the given unit ID.

**DressUpModel**

DressUpModel inherits from PlayerModel and is a specialized version of it that provides functions to preview items.

**Lua Methods**
The DressUpModel type has the following Lua methods.

playerModel:Dress()
Equip the player's default equipment.

playerModel:TryOn(item)
Equip the given item link or item ID.

playerModel:Undress()
Undresses the displayed player.

**TabardModel**

TabardModel inherits from PlayerModel and is a specialized version of it that provides functions to preview guild tabards.

**Lua Methods**
The TabardModel type has the following Lua methods.

tabardModel:CanSaveTabardNow()
Returns 1 if the tabard can be saved.

tabardModel:CycleVariation(variationType, index)
Cycles through the different available options.

tabardModel:GetLowerBackgroundFileName()
Returns the filename of the lower background texture.
tabardModel:GetLowerEmblemFileName()
Returns the filename of the lower emblem texture.

tabardModel:GetLowerEmblemTexture()
Returns the lower emblem texture.

tabardModel:GetUpperBackgroundFileName()
Returns the filename of the upper background texture.

tabardModel:GetUpperEmblemFileName()
Returns the filename of the upper emblem texture.

tabardModel:GetUpperEmblemTexture()
Returns the upper emblem texture.

tabardModel:InitializeTabardColors()
Initializes the tabard frame.

tabardModel:Save()
Saves the tabard.

**ScrollFrame**
ScrollFrame inherits from Frame and represents a scroll bar that can be used to scroll another frame.

**XML Elements**
The ScrollFrame type has only one XML element.

**ScrollChild**
An arbitrary frame that is scrolled by the scroll bar.

**Lua Methods**
The ScrollFrame type has the following Lua methods.

scrollFrame:GetHorizontalScroll()
Returns the current scroll offset.
scrollFrame:GetHorizontalScrollRange()
Returns the maximum scroll range.

scrollFrame:GetScrollChild()
Returns the frame that is being scrolled by the scroll frame.

scrollFrame:GetVerticalScroll()
Returns the vertical scroll offset.

scrollFrame:GetVerticalScrollRange()
Returns the maximum vertical scroll range.

scrollFrame:SetHorizontalScroll(offset)
Sets the current horizontal scroll position.

scrollFrame:SetScrollChild(frame)
Sets frame as the scroll child, that is, the frame that is being scrolled.

scrollFrame:SetVerticalScroll(offset)
Sets the current vertical scroll position.

scrollFrame:UpdateScrollChildRect()
Updates the displayed area of the scroll child. This method is called automatically when the user scrolls in your frame.

Script Handlers
The ScrollFrame type has the following script handlers.

   OnHorizontalScroll(self, offset)
   OnScrollRangeChanged(self, xRange, yRange)
   OnVerticalScroll(self, offset)

ScrollingMessageFrame
ScrollingMessageFrame is a frame type that is derived from Frame; it can be used to display multiple messages with a scroll bar, as in a chat frame.

ScrollingMessageFrame has all XML attributes and elements and Lua methods of MessageFrame. However, it is technically not a message frame, as scrollingMessageFrame:IsObjectType("Font") returns nil. The methods and attributes of MessageFrame are not listed here.
XML Attributes
The `ScrollingMessageFrame` type has only one XML attribute.

`maxLines`
The maximum number of lines that can be added to the scroll frame. Default: 8.

Lua Methods
The `ScrollingMessageFrame` type has the following Lua methods.

`scrollingMessageFrame:AddMessage(text, r, g, b, id, addToStart)`
Adds a message with the given color and ID. The ID can later be used to change the color of the message with `UpdateColorById`. The `addToStart` argument can be used to insert the message at the opposite side from the default insert position.

`scrollingMessageFrame:AtBottom()`
Determines whether the frame is scrolled to the very bottom.

`scrollingMessageFrame:AtTop()`
Determines whether the frame is currently scrolled to the very top.

`scrollingMessageFrame:GetCurrentLine()`
Returns the line number of the last added message. Lines are zero-based; the first line has the line ID 0. Note that a message can consist of multiple lines.

`scrollingMessageFrame:GetCurrentScroll()`
Returns the number of messages that are currently not visible at the bottom.

`scrollingMessageFrame:GetHyperlinksEnabled()`
Determines whether hyperlinks are enabled or disabled in the scrolling message frame.

`scrollingMessageFrame:GetMaxLines()`
Returns the maximum number of lines that can be displayed.

`scrollingMessageFrame:GetNumLinesDisplayed()`
Returns the number of lines that are currently visible in the frame.

`scrollingMessageFrame:GetNumMessages()`
Returns the number of messages that were added to the scrolling message frame.
scrollingMessageFrame:PageDown()
Scrolls down by one page.

scrollingMessageFrame:PageUp()
Scrolls up by one page.

scrollingMessageFrame:ScrollDown()
Scrolls down.

scrollingMessageFrame:ScrollUp()
Scrolls up.

scrollingMessageFrame:ScrollToBottom()
Scrolls to the bottom.

scrollingMessageFrame:ScrollToTop()
Scrolls to the top.

scrollingMessageFrame:SetHyperlinksEnabled(enabled)
Enables or disables hyperlinks in the frame.

scrollingMessageFrame:SetScrollOffset(offset)
Scrolls to a given position from the bottom. Negative offsets will scroll to a position relative to the top.

scrollingMessageFrame:UpdateColorByID(id, r, g, b)
Changes the color of all messages inserted with the given id.

SimpleHTML

SimpleHTML is a frame type that is derived from Frame; it allows you to display simple HTML. You can read the Wikipedia article http://en.wikipedia.org/wiki/HTML if you are not familiar with HTML. The following elements are allowed in a SimpleHTML frame.

- `<a href="hyperlink">text</a>` to create links. Clicking on the link calls the OnHyperlinkClicked handler with the usual arguments. It is not possible to use URLs here.
- `<br/>` to start a new line.
• `<h1>`, `<h2>`, `<h3>`, and `<p>` for normal text. The attribute `align` can be used to control the text alignment and can be `left`, `right`, or `center`.

• `<img src="texturePath" align="left/right/center" width="x" height="y"/>` to embed images. All attributes except for `src` are optional.

You also need the obligatory `<html>` and `<body>` tags or World of Warcraft won’t recognize your code as HTML. Your text should look like this:

```html
<html>
  <body>
    <!-- your code here -->
  </body>
</html>
```

SimpleHTML has all the methods of `Font` except for `CopyFontObject`. But there is a small difference: all `Font` methods take an optional first parameter that can be either "p", "h1", "h2", or "h3" to change the styles of the different available text types. The default value for this optional parameter is "p".

All XML attributes and elements of `Font` are also available; they control the style of `<p>` tags. The elements `FontStringHeader1 – 3` can be used to set the style of `<h1> – `<h3> tags from XML.

### XML Attributes

The `SimpleHTML` type has the following XML attributes.

- **file**
  
  An HTML file to load.

- **hyperlinkFormat**
  
  Defines how HTML links are displayed; this will be processed by `string.format` with the first argument being the target of your link and the second the text. The default value is "|%s|https://%s|", which means your link is displayed like a regular in-game link.

- **text**
  
  The HTML text to display.

### XML Elements

The `SimpleHTML` type has only one XML element.

- **FontStringHeader1 – 3**
  
  `FontString` objects that are used for `h1 – h3` styled texts.

### Lua Methods

The `SimpleHTML` type has the following Lua methods.
simpleHTML: GetHyperlinkFormat()
Returns the hyperlink format.

simpleHTML: GetHyperlinksEnabled()
Returns whether hyperlinks are enabled.

simpleHTML: SetHyperlinkFormat(fmt)
Sets the hyperlink format.

simpleHTML: SetHyperlinksEnabled(enabled)
Enables or disables hyperlinks.

simpleHTML: SetText(html)
Sets the HTML to display.

**Slider**

Slider inherits from Frame and is a frame type to input numeric values.

**XML Attributes**

The Slider type has the following XML attributes.

**defaultValue**

The default value of the slider. Caution: this triggers the script handler OnValueChanged before OnLoad is called.

**drawLayer**

The layer in which the slider’s textures will be drawn. Default: OVERLAY.

**maxValue**

The maximum value of the slider.

**minValue**

The minimum value of the slider.

**orientation**

HORIZONTAL or VERTICAL (default).
valueStep
The step size.

**XML Elements**
The Slider type has the following XML elements.

**ThumbTexture**
A texture object. This texture is used as a thumb (small draggable dot).

**Lua Methods**
The Slider type has the following Lua methods.

`slider:Disable()`
Disables the slider.

`slider:Enable()`
Enables the slider.

`slider:GetMinMaxValues()`
Returns the minimum and maximum values that can be set by the user.

`slider:GetOrientation()`
Returns the orientation; can be either HORIZONTAL or VERTICAL.

`slider:GetThumbTexture()`
Returns the thumb texture.

`slider:GetValue()`
Returns the current value.

`slider:GetValueStep()`
Returns the step size.

`slider:IsEnabled()`
Returns 1 if the slider is enabled.

`slider:SetMinMaxValues(min, max)`
Sets the minimum and maximum values.
slider: SetOrientation(orientation)
Sets the orientation; can be HORIZONTAL or VERTICAL.

slider: SetThumbTexture(texture)
Sets the thumb texture.

slider: SetValue(val)
Sets the current value.

slider: SetValueStep()
Sets the step size.

Script Handlers
The Slider type has only one script handler.

OnValueChanged(self, value)

StatusBar
StatusBar inherits from Frame and can be used to display progress, like a timer.

XML Attributes
The StatusBar type has the following XML attributes.

defaultValue
The default value of the slider.

drawLayer
The layer in which the bar’s textures will be drawn. Default: ARTWORK.

maxValue
The maximum value of the bar.

minValue
The minimum value of the bar.

orientation
HORIZONTAL or VERTICAL (default).
rotatesTexture
Can be set to true to rotate the texture to fit the status bar. Allows you to use horizontal textures for vertical bars. Default: false.

XML Elements
The StatusBar type has the following XML elements.

BarColor
Controls the color of the status bar with its r, g, b, and a attributes.

BarTexture
A texture object that is used for the status bar.

Lua Methods
The StatusBar type has the following Lua methods.

statusBar:GetMinMaxValues()
Returns the minimum and maximum values of the bar.

statusBar:GetOrientation()
Returns the orientation.

statusBar:GetRotatesTexture()
Returns the rotate texture setting.

statusBar:GetStatusbarTexture()
Returns the texture.

statusBar:GetValue()
Returns the current value.

statusBar:SetMinMaxValues(min, max)
Sets the minimum and maximum values.

statusBar:SetOrientation(orientation)
Sets the orientation to either HORIZONTAL or VERTICAL.

statusBar:SetRotatesTexture()
Changes the texture rotation setting.
statusBar: SetStatusBarColor(r, g, b, a)
Changes the color of the bar.

statusBar: SetStatusBarTexture(texture)
Changes the bar’s texture.

statusBar: SetValue(value)
Sets the value.
This appendix contains all the information you are ever going to need about the combat log and combat-related functions. You can probably find all the information presented here on the Internet (for example on WoWWiki), but it is scattered. It can be useful to have all the combat-related events and units in a single place.

**Combat Log Events**

The two combat log events `COMBAT_LOG_EVENT` and `COMBAT_LOG_EVENT_UNFILTERED` have the same arguments; the only difference is that the unfiltered event is called for all combat log events, while the normal event is fired only for events that pass through your currently active combat log filter. You should always use the unfiltered event in your addon unless you are writing a combat log replacement that uses the combat log filter settings.

These events receive numerous arguments (up to 20) based on the subevent. The basic event arguments are also covered in Chapter 9, which deals with the combat log.

**Standard Arguments**

As listed in Table B-1, there are eight standard arguments, which are always available.

**Table B-1. The First Eight Combat Log Arguments**

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>A timestamp that contains the exact time when the event occurred. This can be used as the second argument to <code>date(fmt, time)</code>.</td>
</tr>
<tr>
<td>event</td>
<td>The subevent.</td>
</tr>
<tr>
<td>sourceGUID</td>
<td>The GUID (globally unique identifier) of the entity that generated the event.</td>
</tr>
<tr>
<td>sourceName</td>
<td>The name of the entity that generated the event. For example, if you cast a spell this is your name.</td>
</tr>
<tr>
<td>sourceFlags</td>
<td>Flags that contain additional information about the entity that generated this event. Table B-11 later in this appendix shows how we can extract information from this bit field.</td>
</tr>
<tr>
<td>destGUID</td>
<td>The GUID of the target.</td>
</tr>
<tr>
<td>destName</td>
<td>The target’s name.</td>
</tr>
<tr>
<td>destFlags</td>
<td>Flags of the target.</td>
</tr>
</tbody>
</table>
Additional Arguments for the Prefixes SPELL_ and RANGE_
Table B-2 lists the arguments appended to all SPELL_* and RANGE_* subevents.

Table B-2. Additional Arguments of SPELL and RANGE Events

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spellId</td>
<td>The ID of the spell. This is the same ID that is also used by many WoW database sites, like Wowhead. You can build a URL with the following pattern to retrieve additional information about a spell ID from such a site: <a href="http://www.wowhead.com/?spell=">http://www.wowhead.com/?spell=</a>&lt;spellId&gt;</td>
</tr>
<tr>
<td>spellName</td>
<td>The name of the spell.</td>
</tr>
<tr>
<td>spellSchool</td>
<td>The school of the spell. This is a bit field.</td>
</tr>
</tbody>
</table>

Additional Argument for the Prefix ENVIRONMENTAL_
The ENVIRONMENTAL_* events define only one additional argument, the string environmentalType. This value is one of the following strings:

- DROWNING
- FALLING
- FATIGUE
- FIRE
- LAVA
- SLIME

Additional Arguments for the Suffix _DAMAGE
All subevents that end in _DAMAGE receive the additional arguments listed in Table B-3.

Table B-3. Additional Arguments of Events with the Suffix _DAMAGE

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>The amount of damage the attack did.</td>
</tr>
<tr>
<td>overkill</td>
<td>The overkill damage if the target died from the attack.</td>
</tr>
<tr>
<td>school</td>
<td>The school of the damage; like the spell’s school, this is a bit field.</td>
</tr>
<tr>
<td>resisted</td>
<td>The amount of damage that was resisted.</td>
</tr>
<tr>
<td>blocked</td>
<td>The amount of blocked damage.</td>
</tr>
<tr>
<td>absorbed</td>
<td>The amount of absorbed damage.</td>
</tr>
<tr>
<td>critical</td>
<td>1 if the hit was critical, nil otherwise.</td>
</tr>
<tr>
<td>glancing</td>
<td>1 if it was a glancing strike, nil otherwise.</td>
</tr>
<tr>
<td>crushing</td>
<td>1 if it was a crushing blow, nil otherwise.</td>
</tr>
</tbody>
</table>
Additional Arguments for the Suffix _HEAL

All subevents that end in _HEAL receive the additional arguments listed in Table B-4.

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>The amount of damage the attack did</td>
</tr>
<tr>
<td>overhealing</td>
<td>The overheal</td>
</tr>
<tr>
<td>critical</td>
<td>1 if the hit was critical, nil otherwise</td>
</tr>
</tbody>
</table>

Table B-4. Additional Arguments of Events with the Suffix _HEAL

Additional Arguments for the Suffix _MISSED

All subevents that end in _MISSED receive the additional arguments listed in Table B-5.

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missType</td>
<td>The reason the spell or attack missed</td>
</tr>
<tr>
<td>amount</td>
<td>The amount of damage that missed</td>
</tr>
</tbody>
</table>

The missType is one of the following strings:

- ABSORB
- BLOCK
- REFLECT
- RESIST (amount is nil if the whole spell was resisted)
- VULNERABILITY (amount will be negative, as a vulnerability results in more damage dealt)

Additional Argument for the Suffix _FAILED

There is only one additional argument for failed spells or attacks: missType. Don’t confuse this suffix with the suffix _MISSED and its argument with the same name. A failed spell was not cast at all, for example because of insufficient mana. A missed spell or attack was cast but resisted or parried, for example.

missType is here already a localized string that is displayed as it is in your UI and not an identifier.

Additional Argument for the Suffix _EXTRA_ATTACKS

There is only one additional argument for actions that cause extra attacks: amount, the number of additional attacks granted. All of the extra attacks will generate the usual SWING_DAMAGE events.
Additional Arguments for the Suffixes _ENERGIZE, _DRAIN, and _LEECH

Subevents that end in _ENERGIZE, _DRAIN, or _LEECH receive the additional arguments listed in Table B-6.

Table B-6. Additional Arguments of Events with the Suffixes _ENERGIZE, _DRAIN, and _LEECH

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>The amount of power gained.</td>
</tr>
<tr>
<td>powerType</td>
<td>The power type (mana, energy, health, and so on).</td>
</tr>
<tr>
<td>extraAmount</td>
<td>Only available for _LEECH. The additional power that was drained from the affected target but was not transferred to the caster (for example, when using Drain Life with full health).</td>
</tr>
</tbody>
</table>

The argument powerType is a number that identifies the power type. There are global variables available that should be used instead of the numeric identifiers as it makes your code more readable and resilient to API changes. Table B-7 lists all global variables with their respective values.

Table B-7. Possible Values for powerType

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPELL_POWER_MANA</td>
<td>0</td>
</tr>
<tr>
<td>SPELL_POWER_RAGE</td>
<td>1</td>
</tr>
<tr>
<td>SPELL_POWER_FOCUS</td>
<td>2</td>
</tr>
<tr>
<td>SPELL_POWER_ENERGY</td>
<td>3</td>
</tr>
<tr>
<td>SPELL_POWER_HAPPINESS</td>
<td>4</td>
</tr>
<tr>
<td>SPELL_POWER_RUNES</td>
<td>5</td>
</tr>
<tr>
<td>SPELL_POWER_RUNIC_POWER</td>
<td>6</td>
</tr>
</tbody>
</table>

Additional Arguments for Dispels and Interrupts

There are four events used when buffs are canceled prematurely: SPELL_DISPEL if it was dispelled, SPELL_DISPEL_FAILED if a display attempt was resisted, SPELL_STOLEN if a buff was stolen (for example by Steal Magic), and SPELL_AURA_BROKEN_SPELL if a damage-sensitive debuff (like Polymorph) was broken by an attack on the target.

The event SPELL_INTERRUPT is fired when a spell is interrupted; it uses the same arguments as the dispel events, so it’s listed here. Table B-8 lists the additional arguments of these events.
Table B-8. Additional Arguments of Dispel and Interrupt Events

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extraSpellId</td>
<td>The ID of the spell that was interrupted or dispelled.</td>
</tr>
<tr>
<td>extraSpellName</td>
<td>The name of the interrupted/dispelled spell.</td>
</tr>
<tr>
<td>extraSchool</td>
<td>The school (bit field) of the interrupted/dispelled spell.</td>
</tr>
<tr>
<td>auraType</td>
<td>Only available for SPELL_DISPEL, SPELL_AURA_BROKEN_SPELL, and SPELL_STOLEN; this value is either BUFF or DEBUFF.</td>
</tr>
</tbody>
</table>

Additional Arguments for Buff Events

There are six buff-related events:

- SPELL_AURA_APPLIED
- SPELL_AURA_REMOVED
- SPELL_AURA_APPLIED_DOSE
- SPELL_AURA_REMOVED_DOSE
- SPELL_AURA_REFRESH
- SPELL_AURA_BROKEN

All of them receive the additional argument auraType, which is either BUFF or DEBUFF. The events ending with _DOSE also receive the argument amount, which indicates the value of buff stacks on the affected unit after the event.

Note that the event SPELL_AURA_BROKEN_SPELL is, despite the name, a dispel event and uses different arguments.

Additional Arguments for Damage Shield Events

There are three events related to damage shields: DAMAGE_SHIELD is fired when an attacker is damaged by the effect of a shield (such as Fire Shield). This event behaves like a SPELL_DAMAGE event and receives the same arguments as such an event. DAMAGE_SHIELD_MISSED uses the same arguments as SPELL_MISSED.

Another shield type is one that splits the damage among multiple persons; it uses the event DAMAGE_SPLIT, which takes the same arguments as DAMAGE_SHIELD.

Additional Arguments for Enchant Events

Applying enchants or temporary effects like poisons or Shaman weapon buffs generates the event ENCHANT_APPLIED. The event ENCHANT_REMOVED is fired when the effect wears off. Both events use the arguments listed in Table B-9.
Table B-9. Additional Arguments of Events with the Prefix ENCHANT

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spellName</td>
<td>The name of the spell or poison used</td>
</tr>
<tr>
<td>itemId</td>
<td>The ID of the targeted item</td>
</tr>
<tr>
<td>itemName</td>
<td>The name of the targeted item</td>
</tr>
</tbody>
</table>

The Spell School Bit Field

The default UI provides bit masks to extract flags from this bit field. Table B-10 shows all global variables that hold bit masks for spell bit fields.

Table B-10. Bit Masks for Spell Schools

<table>
<thead>
<tr>
<th>Variable</th>
<th>Bit Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHOOL_MASK_NONE</td>
<td>0x0</td>
</tr>
<tr>
<td>SCHOOL_MASK_PHYSICAL</td>
<td>0x1</td>
</tr>
<tr>
<td>SCHOOL_MASK_HOLY</td>
<td>0x2</td>
</tr>
<tr>
<td>SCHOOL_MASK_FIRE</td>
<td>0x4</td>
</tr>
<tr>
<td>SCHOOL_MASK_NATURE</td>
<td>0x8</td>
</tr>
<tr>
<td>SCHOOL_MASK_FROST</td>
<td>0x10</td>
</tr>
<tr>
<td>SCHOOL_MASK_SHADOW</td>
<td>0x20</td>
</tr>
<tr>
<td>SCHOOL_MASK_ARCANE</td>
<td>0x40</td>
</tr>
</tbody>
</table>

The Unit Flags Bit Field

Table B-11 shows the global variables that can be used as bit masks on the sourceFlags and destFlags arguments.

Table B-11. Bit Masks for Unit Flags

<table>
<thead>
<tr>
<th>Variable</th>
<th>Bit Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMBATLOG_OBJECT_AFFILIATION_MINE</td>
<td>0x00000001</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_AFFILIATION_PARTY</td>
<td>0x00000002</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_AFFILIATION_RAID</td>
<td>0x00000004</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_AFFILIATION_OUTSIDER</td>
<td>0x00000008</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_REACTION_FRIENDLY</td>
<td>0x00000010</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_REACTION_NEUTRAL</td>
<td>0x00000020</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_REACTION_HOSTILE</td>
<td>0x00000040</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_CONTROL_PLAYER</td>
<td>0x00000100</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_CONTROL_NPC</td>
<td>0x00000200</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_TYPE_PLAYER</td>
<td>0x00000400</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_TYPE_NPC</td>
<td>0x00000800</td>
</tr>
<tr>
<td>COMBATLOG_OBJECT_TYPE_PET</td>
<td>0x00001000</td>
</tr>
</tbody>
</table>
Variable | Bit Mask
---|---
COMBATLOG_OBJECT_TYPE_GUARDIAN | 0x00000200
COMBATLOG_OBJECT_TYPE_OBJECT | 0x00000400
COMBATLOG_OBJECT_TARGET | 0x00010000
COMBATLOG_OBJECT_FOCUS | 0x00020000
COMBATLOG_OBJECT_MAINTANK | 0x00040000
COMBATLOG_OBJECT_MAINASSIST | 0x00080000
COMBATLOG_OBJECT_RAIDTARGET1 | 0x00100000
COMBATLOG_OBJECT_RAIDTARGET2 | 0x00200000
COMBATLOG_OBJECT_RAIDTARGET3 | 0x00400000
COMBATLOG_OBJECT_RAIDTARGET4 | 0x00800000
COMBATLOG_OBJECT_RAIDTARGET5 | 0x01000000
COMBATLOG_OBJECT_RAIDTARGET6 | 0x02000000
COMBATLOG_OBJECT_RAIDTARGET7 | 0x04000000
COMBATLOG_OBJECT_RAIDTARGET8 | 0x08000000
COMBATLOG_OBJECT_NONE | 0x80000000

Unit IDs

A unit ID is a string that identifies a player or NPC. What are called primary unit IDs that consist of single identifier (for example player); secondary (as well as tertiary, and so on) unit IDs can be built by appending suffixes to primary IDs. Table B-12 lists all primary unit IDs.

Table B-12. Primary Unit IDs

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>focus</td>
<td>Your focus target.</td>
</tr>
<tr>
<td>player</td>
<td>You.</td>
</tr>
<tr>
<td>pet</td>
<td>Your pet.</td>
</tr>
<tr>
<td>party(n)</td>
<td>The (n)th party member. Note that you are not your own party member, meaning that you cannot refer to yourself by using such a unit ID. (0 &lt; n &lt; 5).</td>
</tr>
<tr>
<td>party(petn)</td>
<td>The pet of the (n)th party member. (0 &lt; n &lt; 5); you cannot refer to your own pet with this unit ID.</td>
</tr>
<tr>
<td>raid(n)</td>
<td>The (n)th raid member. This can, unlike party(n), be you.</td>
</tr>
<tr>
<td>raid(petn)</td>
<td>The pet of the (n)th raid member. This can, unlike raid(petn), be you.</td>
</tr>
<tr>
<td>arena(n)</td>
<td>The enemy players in arena matches. (0 &lt; n &lt; 6).</td>
</tr>
<tr>
<td>target</td>
<td>Your current target.</td>
</tr>
<tr>
<td>mouseover</td>
<td>The unit your mouse is currently over.</td>
</tr>
<tr>
<td>none</td>
<td>Refers to no unit.</td>
</tr>
<tr>
<td>npc</td>
<td>The NPC you are currently interacting with. Interacting means that you have a window like the flight map or quest dialog of the NPC opened.</td>
</tr>
</tbody>
</table>
The two suffixes target and pet can be appended to any unit ID to refer to the target or pet of a unit. Note that there is no difference between partypetn and partynpet when using the unit ID in a unit function like UnitName(uId). But partypetn is a primary unit ID, while partynpet is secondary. There are some events that are only fired for primary IDs; for example, UNIT_TARGET is fired every time a unit with a primary ID changes its target. You will therefore see the partypetn and raidpetn IDs as arguments in many UNIT_* events.

## Unit Functions

There are several functions available that retrieve information about the unit identified by a given unit ID:

- **UnitAffectingCombat(uId):** Checks if the unit is in combat.
- **UnitArmor(uId):** Returns the armor of the unit ID. This can only be used on yourself.
- **UnitAttackBothHands(uId):** Returns 1 if the unit wields two weapons.
- **UnitAttackPower(uId):** Returns the melee attack power. This can only be used on yourself.
- **UnitAttackSpeed(uId):** Returns the attack speed of the unit in seconds.
- **UnitAura(uId, buffId, filter):** Returns information about a specific buff on the unit. The first buff on a unit has the ID 1, the second 2, and so on. The third operational argument is a string that sets a filter. Possible values for the filter are the following:
  - **CANCELABLE:** Matches buffs that can be canceled.
  - **NOT_CANCELABE:** Matches buffs that cannot be canceled.
  - **HARMFUL:** Matches spells that do harm.
  - **HELPFUL:** Matches helpful spells
  - **PLAYER:** Matches buffs cast by you.
  - **RAID:** Matches buffs that you can cast on other raid members.

You can combine multiple filters in the filter string. The default value for filter is "HELPFUL", meaning that it only returns buffs. For example, "HARMFUL PLAYER" matches all harmful spells that have been cast by you.

The return values of this function are name, rank, icon, count, debuffType, duration, expirationTime, unitCaster, and isStealable.

- **UnitCanAssist(uId1, uId2):** Determines whether a unit can assist another unit.
- **UnitCanAttack(uId1, uId2):** Determines whether a unit can attack another unit.
- **UnitCanCooperate(uId1, uId2):** Determines whether two units can cooperate; that is, whether both units are players of the same faction. The difference between this function and UnitCanAssist is that a player can assist an NPC but not cooperate with one.
- **UnitCharacterPoints(uId):** Returns the number of unspent talent points of a unit.
- **UnitClass(uId):** Returns the localized class of a player followed by the English name of the class in capital letters.
• UnitClassification(uId): Gets the classification of a unit; this can be normal, trivial ("gray" mobs), elite, rare, rareelite, or worldboss.

• UnitCreatureFamily(uId): Gets the creature family of a beast (for example, Cat or Spider).

• UnitCreatureType(uId): Gets the creature type of a unit (for example, Beast or Giant).

• UnitDefense(uId): Gets the defense skill of a unit.

• UnitDetailedThreatSituation(uId1, uId2): Returns the threat situation between two units. The return values are isTank, status, threatpct, rawthreatpct, threatvalue.

• UnitExists(uId): Determines whether a unit ID references an existing unit.

• UnitFactionGroup(uId): Returns the faction of a unit (for example, Sons of Hodir or Alliance).

• UnitGUID(uId): Returns the GUID of a unit.

• UnitHasRelicSlot(uId): Determines whether a unit has a relic slot.

• UnitHealth(uId): Returns the current health of a unit.

• UnitHealthMax(uId): Returns the maximum health of a unit.

• UnitInParty(uId): Determines whether a unit is in your party.

• UnitInRaid(uId): Determines whether a unit is in your raid.

• UnitInRange(uId): Returns 1 if uId is less than 30 yards away from you.

• UnitIsAFK(uId): Determines whether a player is AFK.

• UnitIsCharmed(uId): Checks whether a unit is mind-controlled.

• UnitIsConnected(uId): Checks whether a player is online.

• UnitIsCorpse(uId): Determines whether a unit is a corpse.

• UnitIsDead(uId): Determines whether a unit is dead.

• UnitIsDeadOrGhost(uId): Determines whether a unit is dead or in ghost forms.

• UnitIsDND(uId): Checks whether a unit has DND enabled.

• UnitIsEnemy(uId1, uId2): Checks whether two units can attack each other.

• UnitIsFeignDeath(uId): Determines whether a unit is feigning death. This only works on units in your party or raid.

• UnitIsFriend(uId1, uId2): Returns 1 if two units are friendly towards each other.

• UnitIsGhost(uId): Checks whether a unit is in ghost form.

• UnitIsPVP(uId): Checks whether a unit is flagged for PvP.

• UnitIsPVPFreeForAll(uId): Checks whether a unit is flagged for FFA PvP (for example in the Stranglethorn Arena).

• UnitIsPartyLeader(uId): Determines whether a unit is the leader of a party.

• UnitIsPlayer(uId): Determines whether a unit is a player.
- `UnitIsPlusMob(uId)`: Returns 1 for elite mobs.
- `UnitIsTapped(uId)`: Checks whether a mob is tapped by someone else.
- `UnitIsTappedByPlayer(uId)`: Checks whether a mob is tapped by you.
- `UnitIsTrivial(uId)`: Checks whether a mob is trivial (“gray”).
- `UnitIsUnit(uId1, uId2)`: Checks whether two unit IDs reference the same unit.
- `UnitIsVisible(uId1)`: Checks whether a unit is in your range (about 100 yards).
- `UnitLevel(uId)`: Gets the level of a unit.
- `UnitName(uId)`: Returns the name and server (in battlegrounds) of a player.
- `UnitOnTaxi(uId)`: Checks whether a unit is on a taxi service.
- `UnitPlayerControlled(uId)`: Checks whether a unit is controlled by a player.
- `UnitPlayerOrPetInParty(uId)`: Determines whether the unit is a member of your party or a pet of a party member.
- `UnitPlayerOrPetInRaid(uId)`: Works like `UnitPlayerOrPetInParty` but for raid groups.
- `UnitPower(uId, powerType)`: Gets the current amount of `powerType` (mana, energy, and so on) of a unit. `powerType` uses the same values as mentioned in the combat log event above.
- `UnitPowerMax(uId, powerType)`: Gets the maximum amount of `powerType` of a unit.
- `UnitPowerType(uId)`: Gets the current power type of a unit.
- `UnitRace(uId)`: Gets the localized race of a unit followed by a non-localized capitalized string.
- `UnitRangedAttackPower(uId)`: Returns the ranged attack power. This only works on yourself.
- `UnitReaction(uId1, uId2)`: Determines the reaction of `uId1` towards `uId2`. Possible values are friendly, neutral, and hostile.
- `UnitXP(uId)`: Gets the current XP of `uId1`. This only works on yourself.
- `UnitXPMax(uId)`: Gets the maximum XP of `uId1`. This only works on yourself.
Index

**Numbers and Symbols**

3D application
- creating simple, 400–402
- creating the sky in, 402
- creating trees, 401
\36 (ASCII code for $), 395
\124, Lua escape code, 259
255-character limit, circumventing in macros, 354–355
: (colon) operator, 62, 172
-- (double hyphens), designating comments with, 16
== (equality test operator), 27
== (equality operator), 15
= (equality sign), assigning a value to a variable with, 17
% (modulo operator), used in FindNextPosition method, 217
~= (negation of the equality test) operator, 27
| (pipe symbol), as escape character, 258
/ (slash) commands, creating, 63–65
[] (square brackets), using in strings, 19–20
|| (two pipe symbols), representing normal pipe symbol, 259
/ahwgui, creating, 294
/-channel <channel> command, creating, 292–293
/cm gui command, creating, 305
/cooldownmonitor gui slash command, 308
/script command, creating one-minute timer with, 301
/ (slash) commands, creating, 63–65

**A**

<AbsDimension>, 99
absorbed argument, for events with DAMAGE suffix, 243
ACCEPT global variable, 135
access control list (ACL), 82
Ace3 framework, 283–295
  advanced Hello, World with, 286
  first steps with, 285–286
  simple Hello, World addon with, 283–284
  web site address for information about, 283
Ace3 libraries, embedding, 284
AceAddon-3.0 object, OnInitialize method of, 287
AceAddon object, 290
AceConfig-3.0
  configuration with, 290–295
  web site address for documentation, 295
AceConfig frame, automatically generated, 293
AceConfig table, creating for Hello, World mod, 292–293
AceConsole, / commands with, 286
AceDB-3.0, 286
  creating new with default values, 287–288
  data types, 288–289
  saved variables with, 287–289
AceDB-3.0 API documentation, web site address, 288
AceOptions table, 290
  advanced with groups, 290–292
  available option types for, 295
  building / commands from, 293–295
  code for, 291
  simple, 290
AceOption tables, building GUIs from, 293
Ace project, web site address, 5
acl add and acl remove commands, 87
AddCommunityCard method, 216, 240
AddDoubleLine method, 298
addHighest(n, ...) function, writing, 206
AddMessage method
  of chat frame object, 259
__add metamethod, 170
addon communications
  basics of, 228–230
  building poker game client with, 227–240
  example of concatenation in Lua, 367–370
ADDON LOADED event, calling Poker_TableBrowser.ADDON_LOADED function, 224
addon messages
  function for sending test, 229
  playing around with, 229–230
  receiving, 229
  security and, 230–232
  sending, 228
addons
  available message channels, 228
  displaying chat time stamp using a hook, 158
  naming, 228
  profiling in WoW, 363–364
  that make use of graphs, 165
AddOn Studio IDE, 9
AddText method, 289
adjacency matrix, 166
amount argument, for events with DAMAGE suffix, 243
anchor, function of, 99–100
ANCHOR_CURSOR, using, 74
<Anchor> element, 99
AND binary operator
  for reading a specific bit, 249
  truth table of, 249
and operator, 28, 29
announces and warnings, 313–314
anonymous functions, in Lua, 22
API functions, provided by World of Warcraft, 58
Apress web site address, for book source code download, 11
args field, needed by group type, 290
argument option, string.format function, 52
arguments
  buff events, 459
  DAMAGE SHIELD events, 459
  DISPELS and INTERRUPTS, 458–459
  _DRAIN suffix, 458
  ENCHANT events, 459
  _ENERGIZE suffix, 458
  ENVIRONMENTAL_ prefix, 456
  _EXTRA_ATTACKS suffix, 457
  _FAILED suffix, 457
  _LEECH suffix, 458
  in Lua, 15
  of events with _DAMAGE suffix, 456–457
  of events with _HEAL suffix, 457
  of events with _MISSED suffix, 457
  prefixes SPELL_ and RANGE_, 456
  standard for combat log events, 455–456
arithmetic expressions. See arithmetic operators
arithmetic operators, 26–27
arrays
  created by using tables, 160–161
  understanding, 374–377
  what they look like internally, 374–375
array size, in Lua, 375–377
ARTWORK layer, creating template for entries with, 116–118
Assignments, in Lua, 37–38
associated array, 23
associative array, 23
attackingUnit prototype, 177
Attack(target) method, 173
attributes
  building unit frames with, 326–331
  in XML elements, 94
  of header templates, 331–333
  playing around with for secure action button, 336–338
  vs. methods, 173
auctions
  canceling, 89–90
  ending, 84–86
  placing bids, 86–87
  remote control of, 90–91
  starting, 83–84

B
<Backdrop> element, adding to <Frame> element, 103
Backdrops, adding to frames, 102–103
<BackgroundColor> element, in <Backdrop> element, 102
bar:SetText(text) method, 302
bar object, 302
Battle and Defensive Stance, with Shield Bash, 352
Berserker Stance, use of Pummel in, 352
betting rounds, starting, 213–214
bettingRoundStarted state, 210
bgFile attribute, in <Backdrop> element, 102
bids, placing, 86–87
big O notation, convention used in book, 364–366
binary OR operator, truth table of, 250
binary search algorithm, 365
binary XOR operator, 251–252
bindings, in Lua, 1
binSearch function, 365
bit.band function, 249
bit.bnot(bitfield) function, 251
bit.bor, checking for multiple spell schools with, 254
bit fields, 248–252
limits of in Lua with bitlib, 252
vs. booleans, 252
bitlib, 190-191
functions in, 190
web site for downloading, 190
bit library, 190–191. See also bitlib
loading for testing in Lua interpreter, 249
bit.lshift(bitfield, n) function, creating a bit mask with, 250
bit masks
categories of, 255
creating, 249–250
for spell and damage schools, 252–253
for unit flags, 254–255
bits
reading, 249–250
resetting, 251
setting, 250–251
toggling, 251–252
bitwise operations, how they work, 249–250
bitwise OR operator, for setting bits, 250–251
Blizzard Entertainment
useful programs, tools, and web sites, 4
World of Warcraft published by, 2
Blizzard XML files, checking for templates, 109
blocked argument, for events with DAMAGE suffix, 243
blocks, creating using do end blocks, 40
boolean value, in Lua, 21
border, creating around buttons, 111–112
boss mods. See also Deadly Boss Mods
calling constructor in, 311
creating new, 311–312
localizing, 314
break statement, 36
BroadcastMessage method, called in Move-DealerButton method, 217
btn or mod condition, 352
bubble sort algorithm, 366
buff events, arguments for, 459
bug, fixing for World Map, 273–274
buildGuidTable function, code showing, 247
buttons
adding text to new, 130
creating, 118–121
creating from a template, 114–115
creating more, 128–130
defining in XML, 107
filling, 120–121
in XML, 107–108
simple, 110–111
static popup dialogs and, 128–135
button template, using, 108–109
Button type, 424–428
Lua methods, 425–428
script handlers, 428
XML attributes, 424
XML elements, 424–425
<ButtonText> element, defining text of button with, 108
C
CaduneTreeParameters object, trees created from, 401
CANCEL global variable, 135
Cancel method, for canceling timers, 271–272
Cancel or Close buttons, creating, 145
Captures, example using two, 50
cards
- additions to localization file, 196
- calculating a rating of a hand, 200–202
- code loop that generates all 52 cards, 196
- creating, 196–197
- drawing in game, 216
- extracting and rating the best hands, 199–200
- fields in table, 196
- function comparing values of two cards, 199
- getting, 197–198
- loop for testing correct creation of, 197
- shuffling and drawing, 215–216
- cardsByRank table, creating, 196
- card type scores, 201
- caster’s flags, 257
- CastingBarFrameTemplate, defined by Interface\FrameXML\CastingbarFrame.xml, 263
- cast sequence command, 344–345
- CastSpellByName(“name”) function, 319
- Celes, Waldemar, Lua developed by, 1
- \c escape-sequence, 259
- channel argument, 228
- character classes, examples of usage, 49–50
- characters, with special meaning in Lua, 50
- char data type, 288
- ChatFrame1.AddMessage function, 158
- chat frames
  - and their script handlers, 71–72
  - determining default, 259
  - storage of default, 71
  - using escape sequences in, 258–263
- ChatlinkTooltips
  - creating, 71–74
  - module in action, 69
- chat messages
  - hiding, 91–92
  - sending, 355–356
- CHAT_MSG_ADDON
  - complete event handler, 231
  - event arguments, 229
  - event handler for, 231
- CHAT_MSG_ADDON event, registering, 235
- CHAT_MSG_GUILD, 75
- CHAT_MSG_RAID, 75
- CHAT_MSG_WHISPER event, demonstrating event handler with, 75
- chat time stamps, addon that displays using a hook, 158
- check boxes. See check buttons
- CheckButton, GetChecked() method, 146
- check buttons, for setting permissions, 143–144
- CheckButton type, 428–429
  - Lua methods, 429
  - XML attributes, 428
  - XML elements, 428
- CheckFlagCarrier method, 335
- Child, of frame, 98
- Chunk, in Lua, 57
- clampedToScreen="true" attribute, 106
- class and prototype, interchangeable use of, 175
- class data type, 288
- closing tag, of XML element, 94
- closures, in Lua, 152–154
- collectgarbage(), testing garbage collector with, 188–189
- collectgarbage(option) function, memory usage returned by, 362
- collectgarbage(option, value) function, controlling garbage collector with, 188
- collision, in hash table, 378
- color code, format of, 259
- colored texts, 259–260
- color option type, 295
- color reset code, 259
- ColorSelect type
  - Lua methods, 430–431
  - script handlers, 431
  - XML attributes, 430
- combat log, and unit references, 455–464
- COMBAT LOG EVENT, 455
  - working with, 241–244
- combat log events, 455–461
  - standard arguments, 455–456
- COMBAT LOG EVENTS, working with, 241–244
- COMBAT LOG EVENT UNFILTERED, 455
  - working with, 241–244
- COMBATLOG_OBJECT_RAIDTARGET_MASK variable, 255
INDEX

COMBATLOG_OBJECT_REACTION_HOSTILE mask, 255
COMBATLOG_OBJECT_TYPE_PLAYER mask, 255
combat logs, creating, 244
commands, available for macros, 340–346
comments, in XML, 94
community cards, 239–240
and player cards, 216
compile-time errors, 181
concatenation
  example in addon communication, 367–370
  graph of memory usage, 369
  of strings in Lua, 20
  vs. table.concat function performance, 370
conditions
  adding to macros, 346–353
  available for all secure commands, 346–348
  combining to build complex expressions, 349–351
conflicts, resolving in hash tables, 378
control structures, 29
  in Lua, 33–37
  simple, 351–352
conventions, used in examples, 364–366
CooldownBars.lua
  adding Lua functions to, 266
  building new and improved, 301
  file, 263
CooldownBarTemplate, children of, 265
CooldownMonitor
  adding options to, 304–308
  adding wrapper functions to main file, 300
  basic options, 305
  building, 256–274
  creating folder and TOC file for, 256
  creating XML and Lua files for, 256
  improving with Deadly Bar Timers (DBT), 299–308
  integrating Deadly Bar Timers (DBT) in, 302–303
  option to show test bars, 305
  option to unlock and move bars, 305
  placing functions in to be accessed from outside the table, 258
  using Combat Log to build, 241–274
CooldownMonitor_Anchor frame, 267
CooldownMonitor.lua file
  adding code for status bar timer constructor to, 268–269
  building, 256–274
  creating print local variable in, 260
  detecting spells, 256–258
  CooldownMonitor.StartTimer function, creating, 263, 269–270
  CooldownMonitor.Unlock function, ShowMovableBar called by, 303
CooldownMonitor.xml file
  adding frame to, 267
  code for template, 264–265
Cooldown type, 429–430
  Lua methods, 429
  XML attributes, 429
Core.lua file, creating real addon code in, 284–286
coroutine.create(func) function, 391
coroutine library, in Lua, 390–392
coroutine.resume(co, ...) function, 391
coroutine.running() function, 391
coroutines
  available functions that work with, 391
  creating example of, 391–392
  coroutine.status(co) function, 391
coroutine.wrap(func) function, 391
coroutine.yield(...) function, 391
CPU profiling, in WoW, 363–364
CPU time, measuring amount used by Lua scripts, 362
CreateBar, arguments when creating a bar, 302
createCounter function, 152–153
CreateFlagCarrierButton method, called when Warsong Gulch is joined, 334–335
CreateFrame function, 70
  creating timers from template with, 265
createGetter function, 307
CreateMonster function, 174
createSetter function, 307
CreateVector(x, y, z) function, 170
critical argument, for events with DAMAGE suffix, 243
crushing argument, for events with DAMAGE suffix, 243
currentPlayer attribute, 214
Curse, web site address, 5
Curse.com, free Subversion repositories for
addons from, 400
custom timer, creating, 232
cvar (console variable), tuning options with, 358
cyclic lists, in Lua, 165
cyclic references, 155

D
DAMAGE SHIELD events arguments for, 459
DAMAGE suffix, arguments of events with,
242–243
_DAMAGE suffix, arguments of events with,
456–457
data structures
in Lua, 22
object orientation and, 180–181
using another, 222
using tables to build, 160–168
data types
basic in Lua, 17–26
provided by AceDB-3.0, 288
DBM. See Deadly Boss Mods
DBM-Core addon, 299
DBT. See Deadly Bar Timers (DBT)
DBT options, 305–308
setting, 304
Deadly Bar Timers (DBT)
first steps with, 301–302
improving CooldownMonitor with,
299–308
integrating in CooldownMonitor, 302–303
playing around with, 301
Deadly Boss Mods, 309–317
documentation web site addresses, 305,
308
forum web site address, 300
history of, 309–310
localizing, 314
real-world examples from, 50–51
TOC files, 310–311
UNIT DIED handler, 246
web site address, 275, 300
DeadlyBossMods.com, 5
Deadly Boss Mods’ module, 61–62
dealer button, moving, 216–217
debug library, function available in World of
Warcraft, 54
debugstack() function, 182
debug.traceback() function, 182
Decursive addon, 2
deep copy function, 156
DEFAULT_CHAT_FRAME global variable, 71
default chat frame held in, 259
default UI
digging through, 130–131
extracting, 66–67
defaultValue attribute, for sliders, 141
de Figueiredo, Luiz Henrique, Lua developed
by, 1
delete(list, node) function, using in doubly
linked lists, 164
De Morgan’s Law, 349
desc field, in AceOptions table, 290
description option type, 295
destFlags argument, combat log events, 242
destGUID argument, combat log events, 242
destName argument, combat log events, 242
DevTools, event trace window, 5
Dialogs, static popup, 130–135
DIALOG strata, frameStrata attribute, 104
<DisabledFont> element, setting font used
for different states with, 108
disjunctive normal form, transforming ex-
pressions into, 350–351
DISPELS and INTERRUPTS, arguments for,
458–459
distributive law, 350
DKP auctioneer mod
local functions and their scope, 82
starting auctions, 83–84
using timing library for, 81–92
variables and options, 81–82
DKP / commands, creating simple, 87–89
do-end block, 83
do...end block, 157
dofile("localization.en.lua"), 195
do keyword, 38
double hyphens (--), designating comments
with, 16
doubly linked lists
  creating in Lua, 162–163
  in Lua, 161–165
_DRAIN suffix, arguments for, 458
DressUpModel type, 444
Druid healing spell, macro that casts, 342
dummy entries, creating for testing, 126–127
dummy function, adding one that prints a message to chat, 123
dynamic typing, in Lua, 25–26

edgeFile attribute, in <Backdrop> element, 102
<EdgeSize> element, in <Backdrop> element, 102
editBox:GetText() method, 146
edit boxes
  creating simple, 138–139
  textures and font strings for, 139–140
EditBoxOnEnterPressed event handler, 134
EditBox type, 431–435
  Lua methods, 432–435
  script handlers, 435
  XML attributes, 432
editor:AddText(text) method, using, 394
elements
  adding to another element, 101
  in XML terminology, 94
  selecting in the list, 122–123
elseif-block, condition used by, 352
embedded libraries
  creating your own, 279–283
  problems with, 277
  using in World of Warcraft, 275–276
embeds.xml file, creating, 284, 304
ENCHANT events, arguments for, 459
endAuction function, writing, 84–86
EndCombat method, called on all boss mods, 246
end keyword, 15, 38
_ENERGIZE suffix, arguments for, 458
entry actions, defined by states, 211
environment
  global variables contained in, 184–187
  modifying, 185–186
ENVIRONMENTAL events, 242
ENVIRONMENTAL_ prefix, argument for, 456
environmentType argument, 242
__eq metamethod, 170
equality operator (==), 15
equality sign (=), assigning a value to a variable with, 17
equality test operator (==), 27
equipped condition, use of, 348
error messages
  creating custom, 182–183
  suppressing in macros, 357–358
error(msg, level) function, 183
errors
  catching and handling, 181–183
  handling in Lua, 181–187
escape sequences
  provided by Lua, 20
  using in chat frames, 258–263
event argument,
  combat log events, 242
  CHAT_MSG_ADDON, 229
event handlers, 69. See also script handlers
  basics of, 75
  building, 243–244
  creating a function that acts as, 395–396
  fake events for in Lua, 368–369
  for multiple events, 75–77
  in Warhammer Online, 398–399
  simple in Lua, 367–368
  that do all necessary checks, 235–236
event list, web site address, 398
event listeners, using frames as, 70–71
events, working with, 312–313
event trace window, DevTools, 5
execute type, 295
  in AceOptions table, 290
execution path, 319
exercise, templates, 138
exit actions, defined by states, 211
ExpandUpwards option, function to set, 307
expressions, 26–31
  tables and functions as, 30
external Lua files and XML files, including, 95–96
_EXTRA_ATTACKS suffix, argument for, 457
faction data type, 288
factionrealm data type, 288
_FAILED suffix, argument for, 457
fake events, for event handler in Lua, 368–369
Far Cry, Lua used by, 1
FauxScrollFrame_OnVerticalScroll, calling, 124
FauxScrollFrame_Update, calling, 125–126
files, restarts and editing existing, 63
filterIncoming function, 91
filterOutgoing function, 91
filters, combining multiple in filter string, 462
finalizer metamethod, 387
findHighCard(cards) function, 208
FindNextPosition method, 217
findPair(cards) function, testing, 207
finite-state machines (FSM), 210–211
designing poker game servers as, 209
diagram showing all states as bubbles, 210
FlagCarrier variable, 335
Flash Heal (Rank 11) spell, adding for testing, 257
flyable bug, circumventing, 358–359
folder structure, for WoW addons, 58–59
<FontString> element, 105–106
<FontString name>, adding for text boxes, 140–140
FontString type, Lua methods, 421–422
Font type
  inherits from Object type, 407–409
  Lua methods, 408–409
  XML attributes, 407
  XML elements, 407–408
for-loop, 16, 34–36
  more about generic, 159–160
frame:ClearAllPoints() method, calling to delete all points, 101
<Frame> element
  adding code to to create button, 109
  adding layers to, 104–105
  in <Ui> root element, 95
frame:GetPoint(n) method, 100
frame:IsProtected() method, 325
frame:RegisterForDrag(button) method, 106
frame:SetHeight(height) method, 102
frame:SetHook, 322
frame:SetPoint(point, relativeTo, relativePoint, x, y) method, creating a new anchor with, 100
frame:SetWidth(width) method, 102
frames
  adding capability to display to our Lua file, 136
  basics of, 98–106
  children of, 98
  creating in World of Warcraft, 95–95
  creating normal for new poker mod, 135
  for adding a secure hook to event handler, 322
  movable, 106
  overview of XML file, 146–148
  restrictions on protected, 324–325
  retrieving name of, 95
  saving frame position with, 106
  using as event listeners, 70–71
  using Lua to create, 118–119
  using, 74
frame size
  retrieving, 102
  setting, 101
frame strata, 103–104
  setting from Lua, 104
frameStrata attribute
  of a frame, 103–104
  valid values for, 103
frames XML file, overview of, 146–148
Frame type, 412
  Lua methods, 414–420
  script handlers, 420–421
  XML attributes, 412–413
  XML elements, 413–414
frame type button, 107
frame types
  creating more with options, 135–146
  creating in World of Warcraft, 405–454
FrameXML\Fonts.xml, finding all available font objects in, 105
FrameXML\ChatFrame.lua file, 320
FrameXML_Debug function, 97
FuBar plugin
  adding tooltip to, 298
  creating, 295–299
  web site for downloading, 296
func field, in AceOptions table, 290
function
    in Lua, 14
        that calculates factorial of a number, 15–16
function calls, 31–33
functions, restricted, 320–321
function value, in Lua, 21–22

G

game announcements
    receiving, 235–237
    sending, 233–235
game events, working with, 69–92
games, joining, 211–213, 237–239
GameTooltip global variable, 298
GameTooltip method
    Lua methods, 436–437
    setting tooltip content, 437–439
garbage collector, 187–190
    checking for weak references, 188
    controlling, 188
    incremental garbage collection, 188
    Lua 5.0, 187
    option arguments, 188
    testing, 188–190
__gc finalizer metamethod, in Lua, 390
generic for loops, 159–160
GetAddOnCPUUsage(addon) function, 363
GetAddOnMemoryUsage(addon) function, 364
GetAlpha() method, 63
GetAttribute(attribute), retrieving current value of attribute with, 326
GetCard function, 198
GetEventCPUUsage(event) function, 364
getfenv(func or level) function, 185
GetFrameCPUUsage(frame, includeChildren) function, 364
GetFunctionCPUUsage(func, includeSubs) function, 363
getglobal(name) function, provided by World of Warcraft, 184
getMax(...) function, writing, 206–207
getmetatable, calling on a frame, 172
getmetatable(table) function, 169
GetNextPlayer method, 214
GetNumRaidMembers() function, used by buildGuidTable function, 247
GetPlayerMapPosition(unitId) function, 298
GetRealZoneText() function, 298
GetSpellInfo function, arguments for, 262
getter function, stored in get, 292
GetTime() function, 78
GetType() method, 176
glancing argument, for events with DAMAGE suffix, 243
global cooldown, spells that don’t activate, 342
global data type, 288
global environment, 184–185
global variables
    _G, 184–185
    speed of, 373
    vs. local variables, 372–373
grammatical escape sequences, 262–263
graphs, 165–168
    five nodes connected via 10 edges, 166
group header templates, using, 331–333
    group type, 290, 295
GUIDs, examples, 246–247
GUIDs and units, working with, 244–248
GUILD channel, messages from, 231

H

hand
    calculating rating of, 200–202
    finding and rating, 202–208
    finding pairs, 204–205
    finding straights, 202–204
handler field, of option table, 290
hash code, of object, 378
hash function, 378
hash tables
    basics of, 377–378
    collision in, 378
    creating in Lua, 378–381
    resolving conflicts in, 378
    testing in Lua, 381–382, 390
    understanding, 377–382
    using in Lua, 23
head, doubly linked lists, 161
headers, creating a template for, 112–113
header templates, attributes of, 331–333
header type, 295
Healing Wave, macro that casts, 342
_HEAL suffix, arguments of events with, 457
HelloAce.toc, creating, 283–284
Hello, World
  in Warhammer Online, 396–399
  with SciTE, 394–395
Hello, World mod
creating AceConfig table for, 292–293
  with DBT, 301–302
HelloWorld, 289
  folder, creating in Interface/AddOns
    folder, 58
  function, creating with SciTE, 394
HelloWorld Initialize function, creating in
  Lua file, 398
HelloWorld.lua file, creating, 397–398
HelloWorld.mod file, creating, 397
HelloWorld OnChat function, creating,
  398–399
HelloWorld.toc file, creating, 59–60
hidden attribute, 118
Hide method, calling on a tooltip, 74
high cards, finding, 208
higher-order functions, in Lua, 152
HIGHLIGHT level, 105
<HighlightFont> element, setting font for dif-
  ferent states with, 108
hooking, in Lua, 157–158
hooking secure functions, 321–322
hooksecurefunc(tbl, key, hook) function,
  321–322
Howling Fjord, graph in, 168
Hyperlinks, escape sequence for creating,
  260–261
IconLeft option, 305
IconRight option, 305
Identifiers, examples of valid, 17–17
IDEs (integrated development environ-
  ments), 8–11
Ierusalimschy, Roberto, Lua developed by, 1
if-else-elseif block, available in a macro,
  351–352
if/else control structure, 15
if statements, 33
  working with local variables in, 39
  writing alternative to, 39
if-then-else-end construct, 28
inactive attribute, in OnPlayerJoin method,
  212
inCombat variable, mods stored in, 246
/in command, using in macros, 357
<Include> element, including another XML
  file with, 95
index, providing to read value from a table,
  22
__index metamethod, 181
  and constructors, 172
inheritance
  from a template, 107
  in Lua, 175–177
  multiple, 177–180
inherits attribute, 105
inline functions, 32
input action, 211
input option type, 295
insertAfter(list, node, data) function, in dou-
  bly linked lists, 163
insertAtEnd function, in doubly linked lists,
  164
insertAtStart(list, data) function, in doubly
  linked lists, 164
insertBefore function, in doubly linked lists,
  164
Interface AddOn Kit, web site address, 7
interface API function, 62
Interface\FrameXML\CastingbarFrame.xml,
  CastingBarFrameTemplate defined
  by, 263
Interface\FrameXML\FriendsFrame.xml,
  130–131
Interface\FrameXML\UIPanelTemplates.
  xml, 109
inventory IDs, types of, 345
issecurevariable(tbl, key) function, 321
IsSelected function, 122
  updating return statement in, 128
isVisible method, using to fix bug for World
  Map, 273–274
item IDs, 73
item links, 72–73
item slot IDs, 345–346
iterator, when and how it is called, 159–160

Java, web site address, 97
join requests
  receiving responses, 238–239
  sending, 237–238
JoinTable method, setting curHost and calling SendMessage, 237
justifyH attribute, 118

keybinding option type, 295
key/value pairs, hash table entries as, 377

Layers, in a frame, 104–105
  <Layers> element
    adding for creating a title for our frame, 137
    as container for all layers, 105
    valid values for level attribute of, 105
layers and textures, 103–106
_LEECH suffix, arguments for, 458
length operator #, 21
=len userdate metamethod, 389
level attribute, valid values for, 105
lexical (static) scoping, used by Lua, 152
LibFuBarPlugin-3.0
  embedding, 296–297
  for writing a FuBar plug-in, 296
  using, 297–299
  web site address for downloading, 296
  WowAce.com for documentation, 299
libraries
  building a skeleton addon for, 279–280
  creating our own embedded, 279–282
  embedded in WoW, 275–276
Lua standard, 43–55
problems with embedded, 277
problems with overwriting old versions, 281–282
using, 275–317
versioning, 276–277
LibStub
  downloading latest version of, 277
  embedding, 280
  functions accessible by addons and libraries, 277–278
  source code, 278–279
  using, 277–279
list element, building, 111–128
list entries, creating template for, 116–118
lists
  scroll frames for, 123–128
  selecting elements in, 122–123
  sorting, 121–122
LoadOptions (id) method, 303
loadstring, 282
local functions, defining, 40–41
local variables
  creating, 38–41
  example of, 38–39
  vs. global variables, 372–373
localization file, adding lines to, 115
localizations, adding more languages, 317
localized variables, accessing, 316
logical AND, 349
logical NOT, 349–350
logical operators, 28–29
logical OR, combining two conditions with, 350–351
Lua
  advanced, 151–193
  advantage of for creating frames, 93
  advantages of, 1
  and Lugre, 399–402
  available metamethods provided by, 171
  basics of, 13–16, 55
  by Roberto Ierusalimschy, et. al., 1
  creating named frames in, 95
  developed at Pontifical Catholic University, 1
  expressions in, 26–31
  in Warhammer Online, 396–399
  just-in-time compiler, 2
language written in, 2
measuring CPU time used by a script, 362
measuring memory usage in, 362
measuring performance in, 362–366
more about functions in, 151–158
other uses for, 393–403
setting frame size in, 102
setting the strata from, 104
simple event handler in, 367–368
speed of, 2
strings in, 367–371
table.concat(tbl, delimiter) function in, 370
tips, tricks, and optimization, 361–392
using, 13–14
using in macros, 355–359
variables and data types, 16–26
web site address, 2
web site for apps and games that use, 393
web site for libraries and bindings, 403
Lua code, executing in-game, 62
LuaEdit IDE, 8
LuaForge, bitlib project page, 190
Lua libraries, more about, 190–192
Lua methods
  Button type, 425–428
  CheckButton type, 429
  ColorSelect type, 430–431
  Cooldown type, 429
  DressUpModel type, 444
  EditBox type, 432–435
  FontString type, 421–422
  Font type, 408–409
  Frame type, 414–420
  GameTooltip method, 436–437
  MessageFrame method, 440
  Model type, 441–443
  Object type, 406–407
  PlayerModel type, 443–444
  Region type, 410–412
  ScrollFrame type, 445–446
  ScrollMessageFrame, 447–448
  SimpleHTML type, 449–450
  Slider type, 451–452
  StatusBar type, 453–454
  TabardModel type, 444–445
  Texture type, 423–424
Lua.org/Lua-users.org, 5
Lugre
  Lua and, 399–402
  Lua binding for OGRE 3D, 393
  Main function called by, 400
  setting up, 400
  website address, 399
  wiki website address, 402
M
macro feedback, 353
macros
  adding conditions to, 346–353
  available commands for, 340–346
  circumventing 255-character limit in, 354–355
  for interrupting spells, 352
  in Lua, 339–360
  inserting healing spell of class to use, 341
  suppressing error messages in, 357–358
  that incorporate a logical OR in expressions, 350–351
  timers in, 356–357
  tuning options, 358
  ultimate for lazy healers, 351
  using Lua in, 355–359
Main function, called by Lugre, 400
math.cos function, example that calls, 373
math library, functions available in, 43–45
math.random, 215
math.randomseed, in Lua, 215
math.sin function, example that calls, 373
max field, 291
maxValue attribute, for sliders, 141
memory profiling, in WoW, 364
memory usage
  if something goes wrong, 369–370
  measuring in Lua, 362
merge sort, sorting algorithm, 366
MessageFrame method, 439–440
messages
  receiving addon, 229
  sending addon, 228
message types, needed in poker addon, 233
metamethods, and metatables, 169–171
metatables
  adding to all tables in a given table, 224–225
  attaching to a saved table, 224–225
  and metamethods, 169–171
  basic functions used when working with, 169
  overloading operators, 169–170
metatables and variables, saved, 222–225
methods vs. attributes, 173
min field, 291
minValue attribute, for sliders, 141
MISSED suffix, arguments of events with, 458–459
_MISSED suffix, arguments of events with, 457
missTYPE argument, 457
mixins, 285
Model type
  Lua methods, 441–443
  XML attributes, 441
  XML elements, 441
.mod file
  event handlers available, 398–399
  in Warhammer Online, 397
modulo operator (%), used in FindNextPosition method, 217
monster prototype, creating, 177–179
monsterPrototype table, creating, 172–175
/mount command, code for creating, 359
mouse click, function to fake, 331
mouse wheel, scrolling with, 127
movable attribute, making a frame movable with, 106
movable frames, 106
move command, for moving timers, 272
MoveDealerButton method
to determine next active player, 216
using FindNextPosition method, 216–217
MPQ archives, used by Blizzard games, 58–59
MPQ Editor
  browsing MPQ archives with, 66
  web site address, 7
MPQ viewer, browsing MPQ archives with, 66
msg argument, 228
MSV, validating an XML opened in SciTE, 96
__mul metamethod, 170
multiple inheritance, 107
  in Lua, 177–180
multiselect option type, 295
MyMod.SPELL_CAST_START function, updating, 255

\n, embedding new line characters with, 20
name field, in AceOptions table, 290
namespaces, in Lua, 186–187
nearest neighbor algorithm, 167–168
.NET Framework (LuaInterface), 1
NewAddon method, 285
__newindex method, in Lua, 379
NewLibrary method, 278
newList function, 163
NewPlayer commands, 239
newproxy() function, 392
nextNode() function, using in doubly linked lists, 164
nil type, 18
nodes, in doubly linked lists, 161
normal slash commands, 340
  web site for Blizzard list of available, 340
<NormalFont> element, setting font used for different states with, 108
not operator, 28
NPC and vehicle GUIDs, 245
numbers, operators for, 26
number type, 18–19
NUM_CHAT_WINDOWS global variable, 71

object-oriented programming, using in Lua, 171–172
object orientation
  and data structures, 180–181
  in Lua, 171–181
object-oriented API, basics of, 62–63
object-oriented timing library, 220–221
Object type, 405–407
  Font type inherits from, 407–409
  Lua methods, 406–407
  XML attributes, 405–406
OGRE 3D, Pro OGRE 3D Programming about, 399
OGRE 3D project, web site address, 402
OGRE (Lugre), 3D engine from Lua, 1
OK/Create button, creating, 145–146
OnBeforeSave handler, 395–396
OnClick handler, code behind, 330–331
OnDisable method, 285
OnDragEnd script handler, 106
OnDragStart script handler, 106
adding to <Scripts> element, 273
One-Hand condition, 348
OnEnable method, 285
OnEvent script handler, 95, 74–77
OnHide script handler, stopping the anchor with, 273
OnHyperLinkEnter script handler, 71–72
OnHyperLinkLeave script handler, 71–72
OnInitialize, using Print method in, 285
OnInitialize handler, testing, 398
OnLoad script handler, 95, 328
adding line to to save frame position, 106
adding variables used in, 143
needed to initialize frame, 327
performing player frame tasks in, 328–329
setting font strings for sliders with, 141
OnMessage method, 238–239
OnMobKill handler, 246
OnMouseWheel script handler, adding, 127
OnPlayerJoin method, 211–213
On prefix, using methods with, 211
OnShow element, adding code to, 225
onSpellCast function, 258
creating, 261
new and improved version, 262
OnUpdateFuBarPlugin method, calling, 298
OnUpdateFuBarText method, 298
OnUpdateFuBarTooltip method, called by LibFuBarPlugin-3.0, 298
OnUpdate script handler, 236, 281
and performance, 80–81
in SimpleTimingLib, 220
using to build a timing library, 77–81
OnValid script handler, alternatives to, 96
OnValueChanged handler
adding variables used in, 143
arguments for, 141
for sliders, 141
OnVerticalScroll script handler, invoking, 124
opening tag, of XML element, 94
Open method, of library AceConfigDialog-3.0, 293
operating system library (os), 54–55
operator precedence, examples of, 30
operators, overloading, 169–170
optimization, main rules of, 366
optimizing, SimpleTimingLib, 383–387
OptionalDeps field, for embedded libraries, 280
OptionalDeps TOC attribute, 297
Options, tuning, 358
OptionsCheckButtonTemplate, for creating check buttons, 143–144
OptionsSliderTemplate, using, 141
or operator, 28. See also bitwise OR operator
os.clock() function, measuring CPU time used by a script with, 362
os.date, available format options for, 54
os.date(format, time) function, 157
os.time() function, 157
overkill argument, for events with DAMAGE suffix, 243

P
pairs, finding, 204–205
parent frame, 98
pattern basics, in string library, 48–51
pcall(func, arg1, arg2, ...) function, for catching and handling error messages, 181
performance
measuring in Lua, 362–366
OnUpdate script handler and, 80–81
persistence, 65
pet frames, additional attributes for, 333
pet GUIDs, player and, 245
pets, secure slash commands for, 343
pet suffix, for unit IDs, 248
pizza timer, that comes with Deadly Boss Mods, 357
player and pet GUIDs, 245
player cards and community cards, 216
player frame, creating from template, 328–329
player link, 261
PlayerModel type, Lua methods, 443–444
player object, creating, 208–209
PLAYER_TARGET_CHANGED event, using in target frame, 329
points, 100–101
  defining multiple, 103
  deleting all, 101
  diagrams showing available, 100
poker addon
  creating first part of, 98–106
  creating folder and TOC file for, 98
POKER_BROWSER_TITLE, text attribute set to, 105
PokerClient:JoinTable() method, 236
Poker CreateTable frame, putting code in OnShow element of Scripts element of, 225
Poker_CreateTable frame, 136
Poker_CreateTable, Show() method, 136
PokerFunctions.lua, creating for building auxiliary functions, 195–208
poker game client
  building, 232–240
  building with addon communications, 227–240
PokerLib.RateCards function, finding a hand with, 199–200
poker mod
  adding a backdrop to, 136
  adding a size and an anchor to, 136
  creating our popup for, 134–135
  setting options for new, 135
Poker_TableBrowser frame
  defining, 98
  making visible, 99–103
Poker_TableBrowser.IsSelected, called by SortTables function, 122
Poker_TableBrowser.JoinSelectedTable() function, implementing, 123–123
Poker_TableBrowser.SelectEntry, called by SortTables function, 122
Poker_TableBrowser.Update() method, calling to update buttons, 121
popOrCreateFrame function, 266–267
popups
  attributes and event handlers for, 132–133
  creating more advanced, 132–133
  creating simple, 131–132
positioning. See SetPosition method
powerType argument, possible values for, 458
precedence
  from higher to lower priority, 29
  in Lua, 29–30
precision .n option, in string.format function, 52
prefix, 32
primitive operator, in Lua table, 24
print function, 31
  hooking to add a time stamp, 157–158
Print method, using in OnInitialize, 285
print(os.clock()) function, using to measure speed, 362
priority queue, using as data structure for a scheduler, 222
profile data type, 288
profiling functions, calling to profile your code, 363–364
Pro OGRE 3D Programming, by Gregory Junker, 399
protected frames, restrictions on, 324–325
protected method, Show method as, 324
pseudo random number generators, 215
push operation, in stacks, 161
PvP (Player vs. Player) addons, used by unit flags, 256

Q

Quicksort, for sorting tables, 366
quotation marks (""), strings written in, 19

R

race data type, 288
raid frame mod, 2
raid roles, secure slash commands for, 343
RandomCrap addon, 354
random number generators, 215
INDEX

RANGE event, arguments for, 242
RANGE_prefixes, arguments for, 456
range type, 295
  for a numeric value, 291
rawget function, 381
\lr (color reset code), 259
realm data type, 288
receive function, in simple event handler, 367
ReceiveHit method, 177
ReceiveHit(attacker) method, 173
recursion, 15
  recursive table functions, 155–156
tail calls, 154–155
recursive table functions, 155–156
recycling, tables in Lua, 386–387
Region type
  Lua methods, 410–412
  XML attributes, 409–410
  XML elements, 410
RegisterChatCommand(cmd, func) method, creating a slash command in Ace with, 286
RegisterEventHandler(event, func) function, 398
RegisterEvent method, 77
RegisterForClicks method, 109
RegisterForDrag method, adding to OnLoad script handler, 272
RegisterForDrag method, 327
RegisterUnitWatch(frame) function, calling for target frame, 329
regular expressions, available classes for, 49
relational operators, 26–28
relativePoint attribute, 99
<RelDimension>, 99
remote control, of auctions, 90–91
removeArg(n, ...) function, writing, 205
repeat-until loops, 34
reserved keywords, 16–17
ResetCPUUsage() function, 363
reset option
  for cast sequence command, 344–345
  triggers available for, 344
resisted argument, for events with DAMAGE suffix, 243
response codes, AccessDenied and Server-Full, 238
restricted functions, 319–321
return keyword, 15
return statement, 36
  example of, 276
RGB color model, web site for information about, 259
Rock, FuBar based on, 296
root element, in XML document, 94
runCmd function, 356
Runed Titanium Rod, code behind a small link to, 260
runtime errors, 181

S
say chat, macro that sends two messages to, 339
Say(text) method, 173
Scale option, 307–308
schedule function, 220–221
  for timing library, 77–79
Schedule method, 220
Scheduler, for timing library, 77–79
school argument, for events with DAMAGE suffix, 243
SciTE (Scintilla-based Text Editor), 10
  configuration file, 394
  customized version for writing addons, 8
  customizing with Lua, 393–396
  Hello, World with, 394–395
  validating XML in, 96
  web site for config commands, 394
  web site for documentation, 396
script handlers
  Button type, 428
  chat frames and, 71–72
  ColorSelect type, 431
  EditBox type, 435
  making frames draggable with, 137–138
  of Frame type, 420–421
  ScrollFrame type, 446
  Slider type, 452
<S> element
  adding to $parentTableList, 127–128
  adding to frame for dragging, 106
  frame script handlers contained in, 95
  making frames draggable with, 137–138
  scroll frames, for lists, 123–128
ScrollFrame type
   Lua methods, 445–446
   script handlers, 446
   XML elements, 445
Scrolling, with the mouse wheel, 127
ScrollingMessageFrame, chat frame type, 71
ScrollMessageFrame type, 446–448
   Lua methods, 447–448
   XML attributes, 447
secure action button
   advanced, 335–338
   Warsong Gulch battleground mod, 334–335
SecureActionButtonTemplate, 331
secure action button templates, using, 333–338
secure action button types, all possible values for, 333–334
secure and tainted code, in World of Warcraft, 319–324
secure button
   adding attributes to, 336–338
   XML file for, 336
SecureButton_GetModifiedAttribute helper function, 331
securecall(func, ...) function, 322
secure functions, hooking, 321–322
secure slash commands, 340
   available for pets, 343
   available for raid roles, 343–344
   available miscellaneous, 344
   for your (focus) target, 342–343
   list of available for spells and items, 341
secure state driver, in target frame, 329
secure templates
   list of, 325–326
   using for unit frame mod, 324–333
   working with, 319–338
SecureUnitButton_OnClick function, defined in FrameXML/SecureTemplates.lua, 330–331
SecureUnitButtonTemplate, 324
security and addon messages, 230–232
SelectEntry, 122
Selection, fixing code for, 127–128
select option, IconLeft and IconRight possible combinations, 305
select option type, 295
self.BetTimeout, unscheduling calls to, 219
SendCards method, 216
SendState method, 212
server:Showdown() method, 217–218
Server.lua, for building the poker server, 208–219
server object, creating, 209–210
servers table, that holds all currently running servers, 210
setAttribute(attribute, value) method, setting attributes of secure frames with, 326
setAttribute method, of protected frames, 326
SetDefaultSize method, calling, 294
setfenv(func or level, environment) function, 185
SetFormattedText method, 371
SetFuBarText method, 297–298
SetGeneralLocalization method, 315
setglobal(name, value) function, provided by World of Warcraft, 184
SetHyperlink, calling SetOwner method before calling, 74
setmetatable function, 170
setmetatable(table, metatable) function, 169
SetOwner method, 74
SetPoint method
   anchoring a frame to a nameless frame with, 105
   setting new point with, 101
SetPortraitTexture(texture, unit) function, in player frame, 328
setPosition method, creating, 270
SetScript, 70
SetSlot method, 239
setter function, of option, 291
setter option, for Width option, 307
SetText method, 371
SetTexture, 240
SetUserPlaced method, 267
shared subtables, 155
short-circuited operator, in Lua, 29
#show and #showtooltip commands, 353
showdown state, 217–218
Show method, as protected method, 324
ShowTestBars method, 301
calling, 305
#showtooltip command, 353
showTooltip function, 74
ShuffleCard method, with compare helper function, 215
SimpleHTML type, 448
   Lua methods, 449–450
   XML attributes, 449
   XML elements, 449
SimpleTimingLib
   optimizing, 383–387
   optimizing using another data structure, 383–384
   supporting old, 222
   updating to use LibStub, 279–283
SimpleTimingLib-1.0
   building, 280–281
   TOC file for, 282–283
SimpleTimingLib-1.1, building simple, 384–386
Simple Timing Library 2.0, adding object-orientation to, 220–222
SimpleTimingLibTest, building skeleton add-on for, 279
SimpleTimingLibTest.lua, 282–283
<Size> element, setting frame size with, 101
_size field
   in metatable in Lua, 389
   sizes or offsets, 101–102
sky, creating in 3D application, 402
slash command handlers
   creating, 289
   creating for options, 304–305
slash commands
   available, 340–346
   creating simple DKP, 87–89
   for disabling profiling, 363
   for hunters and warlocks, 341
   for moving timers, 272–273
   in DevTools, 5
   secure and normal, 340–341
   to enable profiling in WoW, 363
sliders
   basics of, 140–141
   for small and big blind settings, 142–143
   setting minimum and maximum values for, 141
Slider type, 450
   Lua methods, 451–452
   script handlers, 452
   XML attributes, 450–451
   XML elements, 451
   sorting, the list, 121–122
   sound files, playing and testing, 59
   sourceFlags argument, combat log events, 242
   sourceGUID argument, combat log events, 242
   sourceName argument, combat log events, 242
   special warnings and announces, 313–314
specWarnSurge object, warning text if Mag’lygos casts Power Surge on you, 247
SPELL_CAST_START subevents, printing to chat frame, 243
SPELL event, arguments for, 242
spellID argument, SPELL event, 242
spell link, 261
spellName argument, SPELL event, 242
SPELL_prefixes, arguments for, 456
spells
   adding new to table, 257
   detecting in CooldownMonitor.lua file, 256–258
   spellSchool argument, SPELL event, 242
   Spell school bit field, bit masks for, 460
   spell schools, 252–256
   and unit flags, 248–256
   checking for multiple, 254
   square brackets ([]), using in strings, 19–20
   stack, overflow of, 154
   stack:New() constructor, 180
   stack overflow error message, 154
   stack pointer, in server object, 209
   stackPointer attribute, of server object, 215
   stacks, in Lua, 161
   stance condition, 348
   StartBettingRound method, 213–214
   StartMoving handler, 327
   startTimer function, constructor for building timer object, 266–268
   StartTime functions, 301
   Statements, in Lua, 31–38
   static popup dialogs, 130–135
   StaticPopup_Show function, 130
static scoping, 151–152
status bar timers
  building, 263–274
  constructor for timer object, 268–270
  template defined by Blizzard, 263
StatusBar type
  Lua methods, 453–454
  XML attributes, 452–453
  XML elements, 453
status updates, receiving, 239–240
stopmacro command
  used by macro, 352
  using #showtooltip with, 353
StopMovingOrSizing handler, 327
straights, finding, 202–204
string, in Lua, 19–21
StringBuilder object, 369
string.format function
  available directives for, 51
  optimizing in WoW, 371
  options available in, 52
  texture added as additional argument to, 262
string.gmatch function, 191–192
string.gmatch(str, pattern) function, 191
string.gsub function, 191–192
string.gsub(str, pattern, repl, n) function, 192
string.join(delimiter, ...) function, in WoW, 370
string.join function, 244, 370
string library, 191–192
  basics, 45–48
string pool, 367
  strings stored in, 21
strings, in Lua, 367–371
string.split command, 87
  for getting used strings back, 237
style field, 292
subevents, building event handlers for, 243–244
Subversion (SVN). See SVN repository superclass, 175
SVN repository, web site address, 400
Swatter, 6
syncHandler function, in simple event handler, 367
syntax errors, handling of in WoW game, 97

T
TabardModel type, Lua methods, 444–445
table browser, designing, 110–111
TableBrowser.lua
  adding code to, 223
  adding event handler to, 224
table.concat function vs. concatenation performance, 370
table.concat(tbl, delimiter) function, in Lua, 370
table.data type, in Lua, 22–25
table library, functions available in World of Warcraft, 53–54
tables
  basic optimizations, 382–383
  optimizing, 382–383
  parts of, 374–382
  problem saving, 223
  recycling in Lua, 386–387
  using to build data structures, 160–168
  using to create arrays, 160–161
  tables and functions, as expressions, 30
  table.sort function
    in Lua, 366
    for sorting table that holds cards, 215
    sorting cards with, 199
  table.sort(t, compare) function, 366
  tab (/t), as delimiter between the prefix and message, 228
tail calls, in Lua, 154–155
taint, how it spreads, 322–324
taint.log file, example of, 323–324
talkingUnit prototype, 177
target and focus target, secure slash commands for, 342–343
target argument, 228
target frame, creating from template, 329–330
TargetOfTarget_Update() function, 324
target option, available for all secure slash commands, 341
target suffix, for unit IDs, 248
templates
  available secure, 325–326
  building, 263–265
  creating buttons from, 114–115
  creating for list entries, 116–118
  creating two frames from, 328–330
defining for button, 108
exercise, 138
for scroll frames, 123–124
using group header, 331–333
working with secure, 319–338
!T escape-sequence, added to onSpellCast function, 262
Test prefix, script for simple, 229
Texas Hold’em
  a crash course in, 198–199
  using advanced Lua to extend addon, 195–226
  web site for Wikipedia article, 198
  writing of example game, 93–149
text attribute, 105
textures, 105, 261–262
  adding by hand using the <Layers> element, 139–140
  escape sequence that represents, 261
  and font strings for edit boxes, 139–140
Texture type
  Lua methods, 423–424
  XML attributes, 422
  XML elements, 422
tight loop, testing with, 372
<TileSize> attribute, in <Backdrop> element, 102
timer broadcast system of DBM, how it performs checks, 231
timer localization, 315
timer object, building, 266	
timers, 314
  canceling, 271–272
  creating from template, 265
  handling, 266–268
  moving, 272–273
  updating, 270–271
  using in macros, 356–357
timestamp argument, combat log events, 242
Timestamp function, writing, 395
timestamps, that display in chat frame, 158
timing library
  schedule function for, 77–79
  using for DKP auctioneer mod, 81–92
  using OnUpdate to build, 77–81
TinyPad, web site address, 7
toboolean function, creating, 225
TOC (table of contents) file format, 59–62
TOC files, for Deadly Boss Mods, 310–311
toggle option type, 295
tooltip content, setting, 437–439
tooltips
  adding to FuBar plugin, 298
  using, 73–74
TortoiseSVN (for Windows), web site, 400
tostringall function, 244
tostring(arg) function, 21
transition actions, defined by states, 211
transitions, between finite-state machines
  states, 210
traveling salesman problem, 167–168
trees, creating in 3D application, 401
trinket macro, 357
truth table
  of binary AND operator, 249
  of binary OR operator, 250
  of binary XOR operator, 251	
tuning options, for macros, 358
Two-Hand condition, 348
type field, in AceOptions table, 290
type(x) function, getting type of value with, 18

U

<Ui> element, in World of Warcraft, 94
UIErrorsFrame, macro for clearing, 358
UIFrameFadeOut function, using on the texture flash, 270
UI & Macros forum, web site address, 5
UIPanelButtonTemplate, 109
UIPanelButtonUpTexture global variable, 108
UIParent, 99
UIParent:GetScale() method, retrieving UI scale with, 101
UI scale setting, retrieving, 101
UI (user interface), digging through default, 130–131
unary minus operator, 27
undirected, defined, 166
unified diffs, 4
UNIT DIED handler, Deadly Boss Mods, 246
unit flags and spell schools, 248–256
  bit masks for, 254–255
unit flags bit field, bit masks for, 460–461
unit frame mod, using secure templates for, 324–333
unit frames
  building a template for, 326–328
  building with attributes, 326–331
  using the template built for, 328–330
unit frame template, script handlers defined in, 326–327
unit functions, 462–464
UnitGUID, 247
UnitGUID(unitId) function, 254
UNIT_HEALTH event
  event handler needed for, 327
  needed in player frame, 328–329
  registering, 371
unit IDs
  checking for mouseover, 351
  prefixes for, 247
  primary, 461–462
  suffixes for, 248
UnitName, 247
UnitName(unitId) function, 89
unit references, combat log and, 455–464
unlock command, for moving timers, 272
Unlock function, 301
unpack function, 78
  for returning all cards, 198–198
unschedule function, 221, 385
  for timing library, 79–80
  using in object-oriented timing library, 220
unscheduler, for timing library, 79–80
UpdateAddOnCPUUsage() function, 363
UpdateAddOnMemoryUsage() function, 364
Update function, 125
Update method, for updating timers, 270–271
Upvalues, in Lua, 154, 372
usedSlots attribute, in Lua, 379
userdata metamethods, available in Lua, 389–390
userdata object, using in Lua, 388–390
userdata value, 25

V
values field, 292
valueStep attribute, for sliders, 141
vararg parameter, 41
  using in a table constructor, 43
variable arguments (vararg), 41–43
variables, in Lua, 16–17
variables and metatables, saved, 222–225
vector math, web site for information about, 169
vectors
  function for creating, 169–170
  implementing in a 3-dimensional space, 169
vehicle GUIDs, NPC and, 245
versioning libraries, 276–277
video options, 358
virtual=, 108

W
WaitForBet method, implementing, 218
waitingForBet or showdown state, 210, 214
waitingForBet state, 218–219
waitingForPlayers state, 210
Warhammer Online
  event handlers in, 398–399
  Hello, World in, 396–399
  Lua used by, 1, 393
  .mod file, 397–397
  web site address, 396
Warning localization, 315–316
warnings and announces, 314
Warsong Gulch battleground mod, secure action button, 334–335
weak table, 188
web site addresses
  Ace3 framework information, 283
  AceConfig-3.0 documentation, 295
  AddOn Studio IDE, 10
  API documentation of AceDB-3.0, 288
  author web site, 8
  Blizzard list of available normal slash commands, 340
  book information and revisions, 299
  book source code download, 11
Curse, 5, 400
Deadly Boss Mods, 5, 275, 300
Deadly Boss Mods documentation, 305, 308
detailed diffs for all patches, 67
DevTools, 5
downloading bitlib, 190
downloading FuBar, 296
downloading LibFuBarPlugin-3.0, 296
event list, 398
Interface AddOn Kit, 7
Java, 97
list of applications and games that use Lua, 393
Lua, 2
Lua libraries and bindings, 403
LuaEdit IDE, 8
Lua.org/Lua-users.org, 5
Lugre, 399
MPQ Editor, 7
official Lugre wiki, 402
OGRE 3D project, 402
options available in format strings, 52
RGB color model information, 259
SciTE documentation, 396
SVN repository, 400
Swatter, 6
TinyPad, 7
TortoiseSVN (for Windows), 400
UI & Macros forum, 5
vector math information, 169
Warhammer Online, 396, 399
Wikipedia Texas Hold’em article, 198
wowace.com, 277
WowAce project page, 296
WowAce/Curse, 5
WoWLua, 6
WoW UI Designer, 9
WoWWiki page, 4, 62, 360
weighted, defined, 166
while loops, 33–34
widget or frame functions, 62
Width option, 307–308
World of Warcraft (WOW)
  addon messages provided by, 227–240
  addons, 2–4
  API functions provided by, 58
  available metadata attributes, 60–61
  bitlib in, 190
  CPU profiling in, 363–364
  default UI, 2
  frame types you can create in, 405–454
  heavily modified UI, 2
  in-game tools, 5–7
  Lua used as UI by, 1
  memory profiling in, 364
  optimizing string.format function in, 371
  profiling addons in, 363–364
  running, 57, 58
  secure and tainted code, 319–324
  table.wipe(t) provided by, 386
  using your own metadata attributes, 61
  virtual file system provided by, 58–59
  XML file in, 94
World of Warcraft XML, elements in, 94–96
WoW (World of Warcraft)
  first addon, 58–63
  running a, 57, 58
WowAce.com
  for LibFuBarPlugin-3.0 documentation, 299
  web site address, 277
WowAce project page, web site address, 296
WowAce/Curse, web site address, 5
WoW addons
  creating our first, 58–63
  defined, 2–4
Wow API
  creating, 57, 67
  using, 62–63
Wowhead, looking up the spell on, 244
WoWLua, in-game script editor, 6
WoW UI Designer, 9
WoWWiki, table of useful pages, 4
WoWWiki page, web site address, 62, 360
wrapper function, 43
wstrings, 399
wxWidgets (wxLua), 1
XML

basics of, 94–96
comments in, 94
creating a secure action button with, 336–338
debugging, 97–98
short overview of, 115–116
showing file that includes Lua file and XML file, 95
validating and debugging, 96–98
validating in SciTE, 96
XML and frames, using, 93–149
XML attributes
  Button type, 424
  CheckButton type, 428
  ColorSelect type, 430
  CoolDown type, 429
  EditBox type, 432
  FontString type, 421
  Font type, 407
  Frame type, 412–413
  MessageFrame method, 439
  Model type, 441
  Object type, 405–406
  Region type, 409–410
  ScrollMessageFrame type, 447
  SimpleHTML type, 449
  Slider type, 450–451
  StatusBar type, 452–453
  Texture type, 422
XML document, root element in, 94
XML elements, 94
  Button type, 424–425
  CheckButton type, 428
  EditBox type, 432, 440
  Font type, 407–408
  Frame type, 413–414
  MessageFrame method, 440
  Model type, 441
  Region type, 410
  ScrollFrame type, 445
  SimpleHTML type, 449
  Slider type, 451
  StatusBar type, 453
  Texture type, 422
XML file, in World of Warcraft, 94
XML-IDs, advantage of, 119
XML terminology, 94
XOR operator. See binary XOR operator
xpcall(func, errorHandler) function, for catching/handling errors, 182
<XTexture> elements, defining textures with, 108