

# *CLARION 5*

# **Internet Builder Class Reference**

**COPYRIGHT 1997, 1998 by TopSpeed Corporation**  
**All rights reserved.**

This publication is protected by copyright and all rights are reserved by TopSpeed Corporation. It may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from TopSpeed Corporation.

This publication supports Clarion Internet Connect. It is possible that it may contain technical or typographical errors. TopSpeed Corporation provides this publication “as is,” without warranty of any kind, either expressed or implied.

**TopSpeed Corporation**  
150 East Sample Road  
Pompano Beach, Florida 33064  
(954) 785-4555

**Trademark Acknowledgements:**

TopSpeed® is a registered trademark of TopSpeed Corporation.

Btrieve® is a registered trademark of Pervasive Software.

Microsoft® Windows® and Visual Basic® are registered trademarks of Microsoft Corporation.

All other products and company names are trademarks of their respective owners.

# ***CONTENTS SUMMARY***

<b>FOREWORD</b>	<b>27</b>
<b>INTRODUCTION</b>	<b>29</b>
<b>BROKER CLASS</b>	<b>39</b>
<b>WEB SERVER CLASSES</b>	<b>61</b>
<b>WEB CLIENT MANAGER CLASS</b>	<b>91</b>
<b>BROWSER MANAGER CLASS</b>	<b>107</b>
<b>WEB FRAME CLASS</b>	<b>115</b>
<b>WEB WINDOW CLASSES</b>	<b>127</b>
<b>WEB CONTROL CLASS</b>	<b>217</b>
<b>WEBCONTROLCLASS DERIVED CLASSES</b>	<b>263</b>
<b>WEB AREA CLASSES</b>	<b>407</b>
<b>WEB REPORT CLASS</b>	<b>441</b>
<b>JSL MANAGER CLASS</b>	<b>449</b>
<b>JSL EVENTS CLASS</b>	<b>475</b>
<b>WEB FILES CLASS</b>	<b>487</b>
<b>LAYOUT HTML CLASS</b>	<b>507</b>
<b>SUBMIT ITEM CLASS</b>	<b>519</b>
<b>HTML CLASS</b>	<b>525</b>
<b>TEXT OUTPUT CLASS</b>	<b>569</b>
<b>HTTP CLASSES</b>	<b>579</b>
<b>INDEX</b>	<b>621</b>

# CONTENTS

<b>FOREWORD</b>	<b>27</b>
<b>INTRODUCTION</b>	<b>29</b>
<b>About This Book</b>	<b>30</b>
<b>Clarion Internet Builder Classes</b>	<b>31</b>
Class Libraries Generally .....	31
Internet Builder Classes—The IBC Library .....	31
Using the IBC Library .....	32
Internet Connect Terms and Concepts .....	32
Internet Builder Class Synopsis .....	35
Internet Builder Class Header Files .....	36
Internet Connect Templates and the IBC Library .....	36
<b>Documentation Conventions</b>	<b>37</b>
Reference Item Formats and Syntax Diagrams .....	37
Property (short description of intended use) .....	37
Method (short description of what the method does) .....	38
<b>BROKER CLASS</b>	<b>39</b>
<b>Overview</b>	<b>41</b>
BrokerClass Concepts .....	41
Relationship to Other Internet Builder Classes .....	41
Internet Connect Template Implementation .....	42
Source Files .....	42
Conceptual Example .....	43
<b>BrokerClass Properties</b>	<b>44</b>
Client (WebClientManagerClass object) .....	44
Http (HttpClass object) .....	44
Files (WebFilesClass object) .....	45
ServerName (server identifier) .....	45
<b>BrokerClass Methods</b>	<b>46</b>
Functional Organization—Expected Use .....	46
CloseChannel (close channel to application broker) .....	47
GetAuthorizedInfo (get client password) .....	48
GetClient (return WebClientManagerClass object) .....	49

GetRequestArguments (return browser request) .....	50
Init (initialize the BrokerClass object) .....	51
Kill (shut down the BrokerClass object) .....	52
OpenChannel (open channel to application broker) .....	53
ProcessHTTPHeader (process incoming http) .....	54
SetClient (set client information) .....	55
SetClientBrowser (set browser information) .....	55
TakeFile (send HTML code or JSL data) .....	56
TakeHtmlPage (prepare and send HTML code) .....	57
TakeJslData (prepare and send JSL data) .....	58
TakeUnauthorized (prepare and send access denied page) .....	59

## **WEB SERVER CLASSES 61**

<b>Overview</b> .....	<b>63</b>
WebServerClass Concepts .....	63
ShutDownClass Concepts .....	63
Relationship to Other Internet Builder Classes .....	63
Internet Connect Template Implementation .....	64
Source Files .....	64
Conceptual Example .....	65

<b>WebServerClass Properties</b> .....	<b>66</b>
Active (server communicates with broker) .....	66
ArgIndex (index to browser request string) .....	66
Broker (BrokerClass object) .....	66
CurSubmit (SubmitItemClass object) .....	67
Client (WebClientManagerClass object) .....	67
CommandLine (command line parameters) .....	68
DialogPageBackColor (MESSAGE page background color) .....	68
DialogPageImage (MESSAGE page wallpaper) .....	68
DialogWinBackColor (MESSAGE window background color) .....	68
DialogWinImage (MESSAGE window wallpaper) .....	69
Files (WebFilesClass object) .....	69
GotCommandLine (command line arguments set flag) .....	69
JavaLibraryPath (Java Support Library location) .....	70
PageToReturnTo (return URL) .....	70
ProgramName (Server pathname) .....	71
Timeout (period of inactivity after which to shut down) .....	71

<b>WebServerClass Methods</b>	<b>72</b>
Functional Organization—Expected Use .....	72
Connect (establish communication with Application Broker) .....	74
GetInternetEnabled (return Web/Windows mode) .....	75
GetReadyForPage (return ready-to-continue flag) .....	76
GetRequestedWholePage (return scope of browser request) .....	77
GetSendWholePage (return full or partial refresh flag) .....	78
Halt (immediately shut down the Server) .....	79
Init (initialize the WebServerClass object) .....	80
Kill (shut down the WebServerClass object) .....	81
Quit (shut down the Server normally) .....	82
SetDialogPageBackground (set MESSAGE Web page background) .....	83
SetDialogWinBackground (set MESSAGE Web page window background) .....	84
SetSendWholePage (force full page refresh) .....	85
SetNewPageDisable (suppress outgoing Web pages) .....	86
SetNextAction (return Web page field information) .....	87
TakeEvent (process WebServerClass object events) .....	88
TakePageSent (prepare WebServerClass object for next page) .....	89
<b>ShutDownClass Methods</b>	<b>90</b>
Close (a virtual shut down the Web-enabled application) .....	90
<b>WEB CLIENT MANAGER CLASS</b>	<b>91</b>
<b>Overview</b>	<b>93</b>
WebClientManagerClass Concepts .....	93
Relationship to Other Internet Builder Classes .....	93
Internet Connect Template Implementation .....	93
Source Files .....	93
<b>WebClientManagerClass Properties</b>	<b>94</b>
Broker (BrokerClass object) .....	94
Browser (BrowserManagerClass object) .....	94
IP (client IP address) .....	95
Jsl (JslManagerClass object) .....	95
<b>WebClientManagerClass Methods</b>	<b>96</b>
Functional Organization—Expected Use .....	96
Freq2Id (return HTML control Id) .....	97
Init (initialize the WebClientManagerClass object) .....	98
Kill (shut down the WebClientManagerClass object) .....	99
NextHtmlPage (prepare for next HTML page) .....	100

TakeFile (send HTML code or JSL data) .....	100
TakeHtmlPage (send HTML code) .....	101
TakeJslData (send JSL data) .....	103
TakeUnauthorized (send access denied page) .....	105
<b>BROWSER MANAGER CLASS</b>	<b>107</b>
<b>Overview</b>	<b>109</b>
BrowserManagerClass Concepts .....	109
Relationship to Other Internet Builder Classes .....	109
Internet Connect Template Implementation .....	109
Source Files .....	109
<b>BrowserManagerClass Properties</b>	<b>110</b>
Kind (browser type) .....	110
SetNoCache (external cache control support) .....	110
SupportsStyleSheets (style sheet support) .....	111
SubmitFromJava (applet communication mode) .....	111
<b>BrowserManagerClass Methods</b>	<b>112</b>
Init (initialize the BrowserManagerClass object) .....	112
<b>WEB FRAME CLASS</b>	<b>115</b>
<b>Overview</b>	<b>117</b>
WebFrameClass Concepts .....	117
Relationship to Other Internet Builder Classes .....	117
Internet Connect Template Implementation .....	117
Source Files .....	117
Conceptual Example .....	118
<b>WebFrameClass Properties</b>	<b>119</b>
FrameWindow (MDI frame window) .....	119
MenuBarFeq (menubar control number) .....	119
ToolBarFeq (Toolbar control number) .....	119
<b>WebFrameClass Methods</b>	<b>120</b>
CopyControlsToWindow (a virtual to copy frame controls to child window) ...	120
CopyControlToWindow (copy frame control to child window) .....	122
GetMenuBarFeq (return menubar control number) .....	123
GetToolBarFeq (return Toolbar control number) .....	124
TakeEvent (a virtual to handle menu and toolbar events) .....	125

<b>WEB WINDOW CLASSES</b>	<b>127</b>
<b>Overview</b>	<b>131</b>
WebWindowClass Concepts .....	131
Relationship to Other Internet Builder Classes .....	132
Internet Connect Template Implementation .....	132
Source Files .....	133
<b>WebControlListClass Methods</b>	<b>134</b>
Functional Organization—Expected Use .....	134
AddControlsToLayout (set controls for HTML generation) .....	135
CreateHtml (generate HTML for controls) .....	136
Init (initialize the WebControlListClass object) .....	138
Kill (shut down the WebControlListClass object) .....	138
SetParentDefaults (set a control's children) .....	139
<b>WebWindowBaseClass Properties</b>	<b>140</b>
AllowJava (generate or suppress JavaScript) .....	140
Background (window background color) .....	140
BackImage (window wallpaper) .....	140
BorderWidth (Web page border width) .....	141
CloseImage (close button graphic) .....	141
CreateCaption (include a titlebar on the Web page) .....	141
CreateClose (include a close button on the Web page) .....	142
CreateToolbar (include a toolbar on the Web page) .....	142
DefaultButton (enter key button) .....	143
DefaultButtonNeeded (simulate default button) .....	143
DisabledAction (default HTML for disabled controls) .....	144
Files (WebClientManagerClass object) .....	144
FormatBorderWidth (HTML table cell border width) .....	145
GroupBorderWidth (group box border width) .....	145
HelpDocument (HTML help document) .....	145
HelpEnabled (HTML help enabled flag) .....	146
HelpRelative (remote or local help document) .....	146
HelpStyle (HTML help style) .....	146
HtmlOption (window/control scaling information) .....	147
IsSplash (splash screen flag) .....	147
MenubarFeq (menubar control number) .....	148
MenubarType (menu placement) .....	148
OptionBorderWidth (option box border width) .....	148
PageBackground (web page background color) .....	149

PageImage (web page wallpaper) .....	149
Server (WebServerClass object) .....	149
SheetBorderWidth (sheet border width) .....	150
SnapX (horizontal control alignment factor) .....	150
SnapY (vertical control alignment factor) .....	150
TimerAction (time released browser action) .....	151
TimerDelay (time interval for browser action) .....	151
ToolBarFeq (toolbar control number) .....	152
<b>WebWindowBaseClass Methods</b> .....	<b>153</b>
Functional Organization—Expected Use .....	153
CreateChildHtml (create HTML for control’s children) .....	153
GetBackgroundColor (return Web window background color) .....	154
GetBackgroundImage (return Web window wallpaper) .....	154
GetControl (return control information) .....	155
GetCreateClose (return close button flag) .....	155
GetChildren (return all child controls) .....	156
GetFirstChild (return first child control) .....	157
GetHelpHandler (return HTML to show help document) .....	157
GetHelpReference (return HTML help document reference) .....	158
GetHelpTarget (return HTML HREF for help display) .....	158
GetMenubarFeq (return menubar control number) .....	158
GetPageImage (return Web page wallpaper) .....	159
GetShowMenubar (return menubar include/omit flag) .....	159
GetShowToolBar (return toolbar include/omit flag) .....	159
GetToolBarFeq (return Toolbar control number) .....	160
GetToolBarMode (return toolbar entity) .....	160
GetWebActiveFrame (return WebFrameClass reference) .....	161
<b>WebWindowClass Properties</b> .....	<b>162</b>
Authorize (require username and password) .....	162
AuthorizeArea (name of password protected Web page) .....	163
HtmlTarget (HtmlClass object) .....	163
IsCentered (center or left-justify web window) .....	163
IsSecure (public or secure channel) .....	164
SentHtml (first time process) .....	164
<b>WebWindowClass Methods</b> .....	<b>165</b>
Functional Organization—Expected Use .....	165
AddControl (add control information) .....	167
AddControlsToLayout (set controls for HTML generation) .....	169

BodyFooter (generate HTML BODY footer) .....	170
BodyHeader (generate HTML BODY header) .....	171
CreateChildHtml (create HTML for control's children) .....	172
CreateDummyHtmlPage (write empty Html page) .....	173
CreateHtmlPage (generate HTML for a window) .....	174
CreateJsldata (generate Java Support Library data) .....	175
CreatePageFooter (generate HTML page footer) .....	176
CreatePageHeader (generate HTML page footer) .....	176
CreateUnauthorizedPage (create unauthorized user page) .....	177
GetAuthorized (check user authorization) .....	179
GetBackgroundColor (return Web window background color) .....	180
GetBackgroundImage (return Web page wallpaper) .....	181
GetButtonInClientArea (return button present indicator) .....	182
GetChildren (return all child controls) .....	183
GetControl (return control information) .....	184
GetControlInfo (return control reference) .....	185
GetCreateClose (return close button flag) .....	186
GetFirstChild (return first child control) .....	187
GetHelpHandler (return HTML to show help document) .....	188
GetHelpReference (return HTML help document reference) .....	189
GetHelpTarget (return HTML HREF for help display) .....	190
GetMenubarFeq (return menubar control number) .....	191
GetPageImage (return Web page wallpaper) .....	192
GetShowMenubar (return menubar include/omit flag) .....	192
GetShowToolbar (return toolbar include/omit flag) .....	193
GetTableAttributes (return window HTML <STYLE>) .....	194
GetTargetSecurity (return public or secure flag) .....	195
GetToolbarFeq (return Toolbar control number) .....	196
GetToolbarMode (return toolbar entity) .....	197
GetWebActiveFrame (return WebFrameClass reference) .....	198
Init (initialize the WebWindowClass object) .....	199
Kill (shut down the WebWindowClass object) .....	200
ResetFromControls (set control information) .....	200
SetBackground (set Web page window background) .....	201
SetCentered (center window in Web page) .....	202
SetChildDefaults (set children of each control) .....	202
SetFormatOptions (set Web page scale and alignment) .....	203
SetHelpDocument (enable single document Web page help) .....	204
SetHelpURL (enable multiple document Web page help) .....	205

SetPageBackground (set Web page background) .....	206
SetPassword (require Web page password) .....	207
SetSplash (make this a splash window) .....	208
SetTimer (set Web page timer and action) .....	209
SuppressControl (omit control from Web page) .....	210
TakeCreatePage (fill Client request for page) .....	211
TakeEvent (handle browser and ACCEPT loop events) .....	212
TakeRequest (process browser event/request) .....	213
TakeUnknownSubmit (a virtual to handle unexpected requests) .....	214
TitleContents (set browser titlebar) .....	215
ValidatePassword (verify password) .....	216

## **WEB CONTROL CLASS 217**

### **Overview 219**

WebControlClass Concepts .....	219
Relationship to Other Internet Builder Classes .....	219
Internet Connect Template Implementation .....	220
Source Files .....	220

### **WebControlClass Properties 221**

ActionOnAccept (browser action for control) .....	221
Container (container control) .....	222
DisabledAction (HTML for disabled control) .....	222
Feq (control number) .....	223
IsDynamic (memory allocated flag) .....	223
OwnerWindow (owner window) .....	223
ParentFeq (Web page parent control ) .....	224
RealParentFeq (window parent control) .....	224

### **WebControlClass Methods 225**

Functional Organization—Expected Use .....	225
BeforeResetControl (a virtual for control preprocessing) .....	227
CreateCellContents (write HTML for control) .....	227
CreateCellFooter (end HTML for control attributes) .....	228
CreateCellHeader (begin HTML for control attributes) .....	228
CreateColorParameters (write Java applet color parameters) .....	229
CreateForeColorParameter (write Java applet foreground color parameter) .....	230
CreateHtml (write HTML for control and its attributes) .....	231
CreateHtmlExtra (write HTML for related control) .....	232
CreateJsldata (update Web page controls) .....	232

CreateParams (write all parameters for Java control) .....	233
DoSetChildDefaults (set nested child controls) .....	233
GetAlignText (return text alignment information) .....	234
GetAppletType (return applet type) .....	234
GetBackgroundColor (return background color) .....	235
GetCanDisable (return disable-ability flag) .....	236
GetCellAttributes (return control attributes) .....	237
GetChoiceChanged (return selection change) .....	238
GetEventAction (return browser action) .....	239
GetFont (add font information) .....	240
GetHasHotkey (control text contains '&') .....	240
GetIsChild (return family identity) .....	241
GetLevel (return nesting level) .....	242
GetNameAttribute (return HTML control name) .....	242
GetParentBackgroundColor (return parent background color) .....	243
GetPosition (get control coordinates) .....	244
GetQuotedText (return control text in quotes) .....	245
GetTableAttributes (return HTML STYLE) .....	245
GetText (return control text) .....	245
GetUseChanged (return contents changed indicator) .....	246
GetVisible (return control status flag) .....	247
Init (initialize the WebControlClass object) .....	248
Kill (shut down the WebControlClass object) .....	249
PopFont (restore pre-PushFont font) .....	250
PushFont (implement control font) .....	251
RefreshDisabled (update Web page control status) .....	252
ResetControl (update server control) .....	253
ResetFromQueue (record changes to Server LIST queue) .....	254
SetAutoSpotLink (a virtual to set AutoSpotLink) .....	255
SetBorderWidth (a virtual to set BorderWidth) .....	255
SetBreakable (allow word wrap) .....	256
SetChildDefaults (a virtual to set Web page control children) .....	256
SetDescription (a virtual to set AltText) .....	257
SetEventAction (associate browser action with control event) .....	258
SetParentDefaults (confirm parent) .....	260
SetQueue (a virtual to set the FromQ property) .....	261
UpdateCopyChoice (save selected item number) .....	261
UpdateCopyUse (save copy of control contents) .....	262

<b>WebControlClass Derived Classes</b>	<b>263</b>
<b>Overview</b>	<b>269</b>
WebControlClass Concepts .....	269
Classes Derived from WebControlClass .....	269
Relationship to Other Internet Builder Classes .....	269
Internet Connect Template Implementation .....	269
Source Files .....	270
<b>WebHtmlCheckClass Methods</b>	<b>271</b>
BeforeResetControl (control preprocessing) .....	271
CreateCellContents (generate HTML check box) .....	272
CreateJslData (update Web page control) .....	273
GetHasHotkey (control text may contain &) .....	274
ResetControl (update server control) .....	275
<b>WebHtmlEntryClass Methods</b>	<b>276</b>
CreateCellContents (generate HTML entry box) .....	276
CreateJslData (update Web page control) .....	277
ResetControl (update server control) .....	278
<b>WebHtmlGroupClass Properties</b>	<b>279</b>
BorderWidth (Web page control border width) .....	279
<b>WebHtmlGroupClass Methods</b>	<b>280</b>
CreateHtml (write HTML for GROUP control) .....	280
GetHasHotkey (control text may contain &) .....	281
Init (initialize WebHtmlGroupClass object) .....	282
SetBorderWidth (set Web page control border width) .....	283
<b>WebHtmlItemClass Methods</b>	<b>284</b>
CreateCellContents (generate HTML menu item) .....	284
GetVisible (return control status flag) .....	285
ResetControl (update server control) .....	286
<b>WebHtmlMenuClass Methods</b>	<b>287</b>
CreateCellContents (generate HTML menu) .....	287
CreateHtml (write HTML for MENU control) .....	288
GetCellAttributes (return control attributes) .....	289
GetVisible (return control status flag) .....	290
<b>WebHtmlOptionClass Properties</b>	<b>291</b>
BorderWidth (Web page control border width) .....	291

<b>WebHtmlOptionClass Methods</b>	<b>292</b>
CreateHtml (write HTML for OPTION control) .....	292
CreateJsldata (update Web page control) .....	293
GetTableAttributes (return HTML STYLE).....	294
Init (initialize WebHtmlOptionClass object) .....	295
ResetControl (update server control).....	296
SetBorderWidth (set Web page control border width) .....	297
<b>WebHtmlPromptClass Methods</b>	<b>298</b>
CreateCellContents (generate HTML prompt).....	298
GetHasHotkey (control text may contain &).....	299
<b>WebHtmlRadioClass Methods</b>	<b>300</b>
CreateCellContents (generate HTML radio button) .....	300
GetHasHotkey (control text may contain &).....	301
<b>WebHtmlRegionClass Methods</b>	<b>302</b>
CreateHtmlExtra (write HTML for related control) .....	302
SetParentDefaults (confirm parent) .....	303
<b>WebHtmlSheetClass Properties</b>	<b>304</b>
BorderWidth (Web page control border width) .....	304
<b>WebHtmlSheetClass Methods</b>	<b>305</b>
CreateHtml (write HTML for SHEETcontrol).....	305
CreateTabControl (write HTML for SHEET TABs) .....	307
GetIsChild (return family identity).....	308
GetTableAttributes (return HTML STYLE).....	309
Init (initialize WebHtmlSheetClass object) .....	310
ResetControl (update server control).....	311
SetBorderWidth (set Web page control border width) .....	312
SetChildDefaults (set nested child controls) .....	313
<b>WebHtmlTabClass Properties</b>	<b>314</b>
IsEnabled (control enabled flag) .....	314
<b>WebHtmlTabClass Methods</b>	<b>315</b>
CreateHtml (write HTML for control and its attributes).....	315
CreateJsldata (update Web page control) .....	316
CreateParams (write all tab parameters) .....	317
GetAppletType (return applet type).....	318
GetHasHotkey (control text may contain &).....	319
GetIsChild (return family identity).....	320
GetPosition (get control coordinates) .....	321
GetVisible (return control status flag) .....	322

ResetControl (update server control) .....	323
SetParentDefaults (confirm parent) .....	324
<b>WebHtmlTextClass Methods</b>	<b>325</b>
CreateCellContents (generate HTML text box) .....	325
CreateJsldata (update Web page control) .....	326
ResetControl (update server control) .....	327
<b>WebButtonClass Methods</b>	<b>328</b>
BeforeResetControl (control preprocessing) .....	328
GetHasHotkey (control text may contain &) .....	329
GetVisible (return control status flag) .....	330
ResetControl (update server control) .....	331
<b>WebHtmlButtonClass Methods</b>	<b>332</b>
CreateCellContents (generate HTML button) .....	332
Init (initialize WebButtonClass object) .....	333
ResetControl (update server control) .....	334
<b>WebJavaButtonClass Properties</b>	<b>335</b>
IsEnabled (control enabled flag) .....	335
<b>WebJavaButtonClass Methods</b>	<b>336</b>
CreateHtml (write HTML for control and its attributes) .....	336
CreateJsldata (update Web page control) .....	338
CreateParams (write all button parameters) .....	339
GetAppletType (return applet type) .....	339
GetCanDisable (return disable-ability flag) .....	340
GetFilename (return image filename) .....	341
ResetControl (update server control) .....	342
<b>WebJavaToolButtonClass Methods</b>	<b>343</b>
GetEventAction (return browser action) .....	343
<b>WebImageClass</b>	<b>345</b>
<b>WebHtmlImageClass Properties</b>	<b>346</b>
AltText (text to substitute for image) .....	346
<b>WebHtmlImageClass Methods</b>	<b>347</b>
CreateCellContents (generate HTML image control) .....	347
SetChildDefaults (set image/region relationship) .....	348
SetDescription (set text to substitute for image) .....	348
<b>WebJavaImageClass Properties</b>	<b>349</b>
Filename (image file filename) .....	349

<b>WebJavaImageClass Methods</b>	<b>350</b>
CreateHtml (write HTML for control and its attributes) .....	350
CreateJsldata (update Web page control) .....	352
CreateParams (write all image parameters) .....	353
GetAppletType (return applet type) .....	354
<b>WebListClass Methods</b>	<b>355</b>
GetBackgroundColor (return background color) .....	355
<b>WebHtmlListClass Methods</b>	<b>356</b>
CreateCellContents (generate HTML list box) .....	356
CreateJsldata (update Web page control) .....	357
ResetControl (update server control) .....	358
<b>WebJavaListClass Properties</b>	<b>359</b>
AutoSpotLink (hypertext links) .....	359
EventActionQ (browser actions for listbox events) .....	359
Format (Web list formatting information) .....	360
FromQ (LIST data source) .....	360
QueueActionQ (Server LIST queue changes) .....	361
Started (Java list applet started) .....	361
<b>WebJavaListClass Methods</b>	<b>362</b>
CreateHtml (write HTML for LIST control) .....	362
CreateJsldata (update Web page control) .....	364
CreateParams (write all list parameters) .....	365
GetAppletType (return applet type) .....	366
GetEventAction (return browser action) .....	367
Init (initialize the WebJavaListClass object) .....	368
Kill (shut down the WebJavaListClass object) .....	369
ResetControl (update server control) .....	370
ResetFromQueue (record changes to Server LIST queue) .....	371
SetAutoSpotLink (set live hypertext links) .....	372
SetDirty (force refresh of Web page list box) .....	372
SetEventActin (associate browser action with control event) .....	373
SetQueue (set the data source queue) .....	375
UpdateState (force refresh of Web page list box) .....	376
<b>WebStringClass Methods</b>	<b>377</b>
SetBreakable (allow word wrap) .....	377
<b>WebHtmlStringClass Methods</b>	<b>378</b>
CreateCellContents (generate HTML text string) .....	378
GetCellAttributes (return control attributes) .....	379

<b>WebJavaStringClass Properties</b>	<b>380</b>
AutoSpotLink (hypertext links) .....	380
LastText (last transmitted value) .....	380
<b>WebJavaStringClass Methods</b>	<b>381</b>
CreateHtml (write HTML for control and its attributes) .....	381
CreateJslData (update Web page control) .....	382
CreateParams (write all string parameters) .....	383
GetAppletType (return applet type) .....	384
Init (initialize WebJavaStringClass object) .....	385
SetAutoSpotLink (set live hypertext links) .....	386
<b>WebCloseButtonClass Properties</b>	<b>387</b>
Height (button height) .....	387
Width (button width) .....	387
X (button horizontal position) .....	387
Y (button vertical position) .....	387
<b>WebCloseButtonClass Methods</b>	<b>388</b>
CreateHtml (write HTML for control and its attributes) .....	388
CreateJslData (update Web page control) .....	390
CreateParams (write all string parameters) .....	391
GetAppletType (return applet type) .....	392
GetCloneFeq (return button to mimic) .....	393
GetPosition (get control coordinates) .....	394
GetVisible (return control status flag) .....	395
Init (initialize WebCloseButtonClass object) .....	396
ResetControl (apply Web page button action) .....	397
<b>WebHotlinkClass Methods</b>	<b>398</b>
CreateCellContents (generate HTML hypertext link) .....	398
<b>WebLiteralClass Properties</b>	<b>399</b>
Text (Web page text) .....	399
<b>WebLiteralClass Methods</b>	<b>400</b>
CreateHtml (write HTML for control) .....	400
GetCellAttributes (return control attributes) .....	401
<b>WebNullControlClass Methods</b>	<b>402</b>
CreateHtml (write HTML for control) .....	402
CreateJslData (update Web page control) .....	402
GetAppletType (return applet type) .....	403
GetCellAttributes (return control attributes) .....	404

GetIsChild (return family identity) .....	405
GetVisible (return control status flag) .....	406

## **WEB AREA CLASSES 407**

### **Overview 409**

WebAreaClass Concepts .....	409
Relationship to Other Internet Builder Classes .....	409
Internet Connect Template Implementation .....	410
Source Files .....	410

### **WebAreaClass Properties 411**

Background (area background color) .....	411
BackImage (area wallpaper) .....	411
LocalFont (area font information) .....	411

### **WebAreaClass Methods 412**

GetBackgroundColor (return background color) .....	412
GetCellAttributes (return area attributes) .....	413
GetFont (add font information) .....	414
GetVisible (return control status flag) .....	415
Init (initialize the WebAreaClass object) .....	416
Kill (shut down the WebAreaClass object) .....	417
PushFont (implement area font) .....	418
SetBackground (set Web page area background) .....	419
SetFont .....	420
SetParentDefaults (confirm parent) .....	421

### **WebCaptionClass Properties 422**

Alignment (text justification) .....	422
--------------------------------------	-----

### **WebCaptionClass Methods 423**

CreateHtml (write HTML for caption and its attributes) .....	423
GetCellAttributes (return caption attributes) .....	424
GetPosition (get control coordinates) .....	425
GetText (return caption text) .....	426
GetVisible (return control status flag) .....	427
Init (initialize the WebCaptionClass object) .....	428

### **WebMenuBarClass Methods 429**

CreateHtml (write HTML for menubar and its children) .....	429
GetPosition (get control coordinates) .....	430
GetVisible (return control status flag) .....	431
Init (initialize the WebMenuBarClass object) .....	432

<b>WebToolBarClass Methods</b>	<b>433</b>
CreateHtml (write HTML for toolbar and its children) .....	433
GetAppletType (return applet type) .....	434
GetPosition (get control coordinates) .....	435
GetVisible (return control status flag) .....	436
<b>WebClientAreaClass Methods</b>	<b>437</b>
CreateHtml (write HTML for client area and its children) .....	437
GetBackgroundColor (return background color) .....	438
GetPosition (get control coordinates) .....	439
<b>WEB REPORT CLASS</b>	<b>441</b>
<b>Overview</b>	<b>443</b>
WebReportClass Concepts .....	443
Relationship to Other Internet Builder Classes .....	443
Internet Connect Template Implementation .....	443
Source Files .....	444
<b>WebReportClass Properties</b>	<b>445</b>
Html (HtmlClass object) .....	445
Q (report image files to preview) .....	445
NumPages (report page count) .....	446
Server (WebServerClass object) .....	446
<b>WebReportClass Methods</b>	<b>447</b>
Init (initialize the WebReportClass object) .....	447
Kill (shut down the WebReportClass object) .....	447
Preview (send report pages to client browser) .....	448
SetNumPages (set maximum report page count) .....	448
<b>JSL MANAGER CLASS</b>	<b>449</b>
<b>Overview</b>	<b>451</b>
JSLManagerClass Concepts .....	451
Relationship to Other Internet Builder Classes .....	452
Internet Connect Template Implementation .....	452
Source Files .....	452
<b>JSLManagerClass Properties</b>	<b>453</b>
Client (WebClientManagerClass object) .....	453
Files (WebFilesClass object) .....	453
Security (secure or public communication) .....	454
Target (TextOutputClass object) .....	454

<b>JSLManagerClass Methods</b>	<b>455</b>
Functional Organization—Expected Use .....	455
AddQueueEntry (add a Java list row) .....	456
BeginUpdate (write JSL data delimiters) .....	457
CloseChannel (close output channel and notify Client) .....	458
Init (initialize the JSLManagerClass object) .....	459
Kill (shut down the JSLManagerClass object) .....	459
OpenChannel (open channel for JSL data) .....	460
RemoveAllQueueEntries (delete all Java list rows) .....	462
RemoveQueueEntries (delete Java list rows) .....	463
ScrollQueueDown (scroll Java list down) .....	464
ScrollQueueUp (scroll Java list up) .....	465
SelectControl (set Java control to update) .....	466
SetAttribute (set Java control attribute—string) .....	467
SetAttributeFilename (set Java control attribute—pathname) .....	468
SetAttributeLong (set Java control attribute—number) .....	469
SetChecked (set Java check box status) .....	470
SetIconAttribute (display icon in Java list) .....	471
SetListChoice (highlight HTML list row) .....	472
SetQueueEntry (replace a Java list row) .....	473
SetValue (set Java control contents) .....	474
 <b>JSL EVENTS CLASS</b>	 <b>475</b>
<b>Overview</b>	<b>477</b>
JsEventsClass Concepts .....	477
Relationship to Other Internet Builder Classes .....	478
Internet Connect Template Implementation .....	478
Source Files .....	478
 <b>JslEventsClass Properties</b>	 <b>479</b>
EventQ (equivalent Clarion and JSL events) .....	479
 <b>JSLEventsClass Methods</b>	 <b>480</b>
Functional Organization—Expected Use .....	480
AddEvent (add a Clarion/JSL event pair) .....	481
Init (initialize the JSLEventsClass object) .....	482
Kill (shut down JSLEventsClass object) .....	483
GetEventNumber (returnClarion event number) .....	484
GetEventString (return event/action pairs) .....	485

<b>WEB FILES CLASS</b>	<b>487</b>
<b>Overview</b>	<b>489</b>
WebFilesClass Concepts .....	489
Relationship to Other Internet Builder Classes .....	489
Internet Connect Template Implementation .....	489
Source Files .....	490
<b>WebFilesClass Methods</b>	<b>491</b>
Functional Organization—Expected Use .....	491
GetAlias (return HTML alias for file) .....	492
GetDirectory (return temporary folder name) .....	493
GetFilename (return temporary file pathname) .....	494
GetProgramRef (return HTML program reference) .....	495
GetPublicDirectory (return public folder name) .....	496
GetRelativeFilename (return filename for Broker) .....	498
GetSeparateSecure (return web server security support) .....	499
GetTempFilename (return filename for Server) .....	500
Init (initialize the WebFilesClass object) .....	501
Kill (shut down the WebFilesClass object) .....	502
LoadImage (return linked image filename) .....	503
RemoveAll (remove temporary files and folders) .....	504
SelectTarget (set public or secure channel) .....	505
<b>LAYOUT HTML CLASS</b>	<b>507</b>
<b>Overview</b>	<b>509</b>
LayoutHtmlClass Concepts .....	509
Relationship to Other Internet Builder Classes .....	509
Internet Connect Template Implementation .....	510
Source Files .....	510
Conceptual Example .....	510
<b>LayoutHtmlClass Properties</b>	<b>511</b>
SnapX (horizontal grid snap) .....	511
SnapY (vertical grid snap) .....	511
Style (HTML table style) .....	511
<b>LayoutHtmlClass Methods</b>	<b>512</b>
Functional Organization—Expected Use .....	512
CreateHtml (generate HTML table) .....	513
Init (initialize the LayoutHtmlClass object) .....	514

Insert (add control to layout process) .....	515
Kill (shut down the LayoutHtmlClass object) .....	516
SetCell (set current control/cell) .....	517

## **SUBMIT ITEM CLASS** **519**

### **Overview** **521**

SubmitItemClass Concepts .....	521
Relationship to Other Internet Builder Classes .....	521
Internet Connect Template Implementation .....	521
Source Files .....	521

### **SubmitItem Properties** **522**

Event (event number) .....	522
Extra (other event information) .....	522
Feq (control number) .....	522
Name (new control contents) .....	523
NewValue (new control contents) .....	523

### **SubmitItem Methods** **524**

Reset (set SubmitItem properties) .....	524
---	-----

## **HTML CLASS** **525**

### **Overview** **527**

HtmlClass Concepts .....	527
Relationship to Other Internet Builder Classes .....	527
Internet Connect Template Implementation .....	527
Source Files .....	528

### **HtmlClass Properties** **529**

AppletCount (Java applets on this HTML page) .....	529
Browser (browser manager object) .....	529
Client (client manager object) .....	529
Files (file manager object) .....	530
FirstControl (first input control) .....	530
FirstSelectable (first input control select all flag) .....	530
JavaLibraryCab (Java Support Library cabinet name) .....	531
JavaLibraryZip (Java Support Library zip name) .....	531
Option (web page scale information) .....	532
UseFonts (use Windows fonts on Web page) .....	532

### **HtmlClass Methods** **533**

Functional Organization—Expected Use .....	533
--	-----

CreateOpen (start new HTML page) .....	535
GetFontChanged (return font changed flag) .....	536
GetFontStyle (return HTML STYLE string) .....	537
GetPixelsX (convert horizontal dialog units to pixels) .....	538
GetPixelsY (convert vertical dialog units to pixels) .....	539
GetControlReference (return control HREF) .....	540
Init (initialize the HtmlClass object) .....	541
Kill (shut down the HtmlClass object) .....	541
PopFont (restore pre-PushFont font) .....	542
PushFont (set current font) .....	543
TakeNewControl (set first control) .....	544
WriteAppletDimParameter (write Java applet dim parameter) .....	545
WriteAppletFilenameParameter (write Java applet filename parameter) .....	546
WriteAppletFontParameter (write Java applet font parameter) .....	547
WriteAppletFooter (end Java applet) .....	548
WriteAppletHeader (begin Java applet) .....	549
WriteAppletOptParameter (write Java applet parameter) .....	550
WriteAppletParameter (write Java applet parameter) .....	551
WriteAppletUAID (write Java applet unique Id) .....	552
WriteChildAppletFooter (end child Java applet) .....	553
WriteChildAppletHeader (begin child Java applet) .....	554
WriteContainerAppletHeader (begin container applet) .....	555
WriteControlFooter (end HTML control) .....	556
WriteControlHeader (begin HTML control) .....	556
WriteEventHandler (write HTML control accepted action) .....	557
WriteFontFooter (end HTML font) .....	559
WriteFontHeader (begin HTML font) .....	559
WriteFormFooter (end HTML FORM) .....	560
WriteFormHeader (begin HTML FORM) .....	560
WriteJavaScript (write shared JavaScript functions) .....	561
WriteOnFocusHandler (write HTML control selected action) .....	562
WriteRefreshTimer (write HTML timer refresh) .....	563
WriteSpace (write HTML space) .....	564
WriteSubmitApplet (write Java applet coordinator) .....	565
WriteTableFooter (end HTML TABLE) .....	566
WriteTableHeader (begin HTML TABLE) .....	566
WriteTableNewCol (begin HTML TABLE CELL) .....	567
WriteTableNewRow (begin HTML TABLE ROW) .....	567
WriteText (write breakable text string) .....	568

<b>TEXT OUTPUT CLASS</b>	<b>569</b>
<b>Overview</b>	<b>571</b>
TextOutPutClass Concepts .....	571
Relationship to Other Internet Builder Classes .....	571
Internet Connect Template Implementation .....	571
Source Files .....	571
<b>TextOutputClass Methods</b>	<b>572</b>
Functional Organization—Expected Use .....	572
Close (close the file) .....	573
CreateOpen (create and open the file) .....	574
GetSize (return file size) .....	575
Open (open the file) .....	576
Write (write text) .....	577
Writeln (write text and newline marker) .....	578
<b>HTTP CLASSES</b>	<b>579</b>
<b>Overview</b>	<b>581</b>
HttpClass Concepts .....	581
HttpPageBaseClass Concepts .....	581
Relationship to Other Internet Builder Classes .....	582
Internet Connect Template Implementation .....	582
Source Files .....	582
<b>HttpClass Properties</b>	<b>583</b>
Arguments (incoming field data) .....	583
BrowserInfo (Web browser properties) .....	584
Cookies (information stored on Client) .....	585
Files (WebFilesClass object) .....	586
HttpPage (HttpPageBaseClass object) .....	586
ProgName (server name) .....	587
ProcName (password protected Web page) .....	587
ServerInfo (outgoing http information) .....	588
<b>HttpClass Methods</b>	<b>589</b>
Functional Organization—Expected Use .....	589
FinishPage (prepend http to outgoing transmission) .....	591
GetArguments (return incoming field data) .....	591
GetAuthorizedInfo (get client password) .....	592
GetBrowserProperty (return browser property) .....	593
GetCookie (return cookie value) .....	594

GetServerProperty (return outgoing http information) .....	595
Init (initialize the HttpClass object) .....	596
Kill (shut down the HttpClass object) .....	596
PreparePage (prime ServerInfo for next transmission) .....	597
PreparePageForBrowser (prime ServerInfo for HTML transmission) .....	598
PreparePageForJava (prime ServerInfo for JSL transmission) .....	599
PrepareUnauthorized (prime ServerInfo for password challenge) .....	600
ProcessHeader (process incoming http) .....	601
SendCookies (update cookies on client) .....	601
SetBrowserProperty (set browser property) .....	602
SetCookie (set cookie value) .....	603
SetProgName (set server name) .....	604
SetProcName (set protected area name) .....	605
SetServerProperty (set outgoing http item) .....	606
<b>HttpPageBaseClass Properties</b> .....	<b>607</b>
ExpireDateTime (transmission expiration timestamp) .....	607
FileHandler (TextOutputClass object) .....	607
FileLen (transmission size) .....	608
GotHtmlBody (http only transmission) .....	608
HtmlFilename (HTML/JSL transmission file) .....	609
Http (HttpClass object) .....	609
NowDateTime (transmission timestamp) .....	610
PageFilename (entire transmission file) .....	610
Status (status of Client request) .....	611
<b>HttpPageBaseClass Methods</b> .....	<b>612</b>
Functional Organization—Expected Use .....	612
AppendDefaultBody (supply default HTML) .....	613
FinishPage (prepend http to outgoing transmission) .....	614
HandleStatusCode (process status code) .....	615
Init (initialize the HttpPageBaseClass object) .....	616
Kill (shut down the HttpPageBaseClass object) .....	617
PreparePage (set outgoing http items) .....	617
PreparePageBody (virtual to set outgoing http items) .....	618
SetupHttpStatus (set outgoing http status indicators) .....	619
WritePageBody (write outgoing http body) .....	620



# ***FOREWORD***

Welcome to the Clarion Internet Builder Class (IBC) Library Reference! This book is designed to be your every day reference to Clarion's IBC Library. Your only contact with the IBC Library may be through Clarion's Internet Templates. However, if you want to dig deeper and tweak your web-enabled applications, or take complete control of the HTML code generation process, then this is the book for you.

Once you've become familiar with the Clarion's Internet Templates through the *Internet Connect User's Guide*, you may need more information on the finer points of the Internet Builder Class properties and methods that the templates rely on.

That's why we created this reference. This reference provides in-depth discussions of the IBC Library. It shows you how the IBC Templates use the powerful IBC Library objects—and how you can use, reuse, and modify the IBC Library within your hand-coded project.



# ***INTRODUCTION***

<b>About This Book</b>	<b>30</b>
<b>Clarion Internet Builder Classes</b>	<b>31</b>
Class Libraries Generally .....	31
Internet Builder Classes—The IBC Library .....	31
Using the IBC Library .....	32
Internet Connect Terms and Concepts .....	32
Internet Builder Class Synopsis .....	35
Internet Builder Class Header Files .....	36
Internet Connect Templates and the IBC Library .....	36
<b>Documentation Conventions</b>	<b>37</b>
Reference Item Formats and Syntax Diagrams .....	37
Property (short description of intended use) .....	37
Method (short description of what the method does) .....	38

## About This Book

The *IBC Library Reference* describes the Internet Builder Class (IBC) Library.

The Internet Builder Classes are designed to “Web-enable” Clarion applications. In effect, the IBC Library turns your Clarion application into a dynamic HTML code generator that duplicates its Windows operation through any Java-enabled Web browser. That’s right; your fully functional Windows application can be run through any Java-enabled browser, including those running on Unix, Sun, O/S2, and Macintosh operating systems.

The *IBC Library Reference* describes how this takes place, by providing an overview of each class or related group of classes. It provides specific information on the public properties and methods of each class, plus examples for using them. It also shows you the source files for each class and describes the relationships between the classes.

# Clarion Internet Builder Classes

## Class Libraries Generally

---

The purpose of a class library in an Object Oriented system is to help programmers work more efficiently by providing a safe, efficient way to reuse pieces of program code. In other words, a class library should relieve programmers of having to write certain routines by letting them use already written generic routines to perform common or repetitive program tasks.

In addition, a class library can reduce the amount of programming required to implement changes to an existing class based program. By deriving classes that *incrementally* add to or subtract from the classes in the library, programmers can accomplish substantial changes without having to rewrite the base classes or the programs that rely on the base classes.

## Internet Builder Classes—The IBC Library

---

### Typical Reusability and Maintenance Benefits

The Internet Builder Classes (IBC Library) provide all the benefits of class libraries in general. Plus, the Internet Connect Templates automatically generate code that uses and reuses the robust, flexible, and solid (pre-tested) objects defined by the IBC Library. Further, the templates are designed to help you easily derive your own classes from the Internet Builder Classes.

Of course, you need not use the templates to use the Internet Builder Classes. However, the template generated code certainly provides appropriate examples for using the IBC Library in hand coded programs. Either way, the bottom line for you is more powerful programs with less coding.

### IBC Library Scope and Purpose

The Internet Builder Classes have a fairly specific focus or scope. That is, *its objects are designed to* “Web-enable” a Clarion application. **In effect, the Internet Builder Classes turn your Clarion application into a dynamic HTML code generator.**

The IBC Library contains objects that

- Communicate with Java-enabled Web browsers through TopSpeed’s Application Broker
- Generate HTML equivalents to application WINDOWS
- Generate JSL data equivalents to application QUEUES, USE variables, etc.

- Accept and process browser generated events and requests

In summary, the IBC Library supports Web-enabled Windows applications.

## Using the IBC Library

---

Typically, the Server program (your Web-enabled application) calls only a very few of the many IBC Library methods. These few methods call all of the other IBC methods needed to accomplish their purpose. See *IBC Library Quick Reference* in the *Internet Connect User's Guide* for more information on this short list of IBC primary interface methods.

## Internet Connect Terms and Concepts

---

### Key Terms

<b>Server</b>	Your Web-enabled Clarion application, which generates HTML code to represent each of your application's windows and reports, and receives and processes Client generated events.
<b>Client</b>	The Java-enabled browser program (such as Internet Explorer, Netscape Navigator, or Communicator), which receives and displays the Server generated HTML, then dispatches new requests.
<b>Broker</b>	TopSpeed's Application Broker, which handles all communications between the Client and the Server. That is, the Client and Server do not directly communicate with each other; each communicates only with the Broker.

### **Java Support Library (JSL)**

TopSpeed provides a Java Support Library with each Internet Connect license. This small library of Java objects supports Windows-like controls for the HTML pages generated by the Server. The library must reside on the Client computer and is automatically downloaded by the Application Broker the first time the Client references a library object. The Library is downloaded as a cabinet (.CAB) file, an archive file (.ZIP), or as individual object files, depending on the Client's browser.

This book uses these terms as defined here. That is, Server always refers to a Web-enabled Clarion application; Client refers to the Java-enabled browser that is controlling the Server; and Broker refers to the Application Broker that routes messages between the Client and Server.

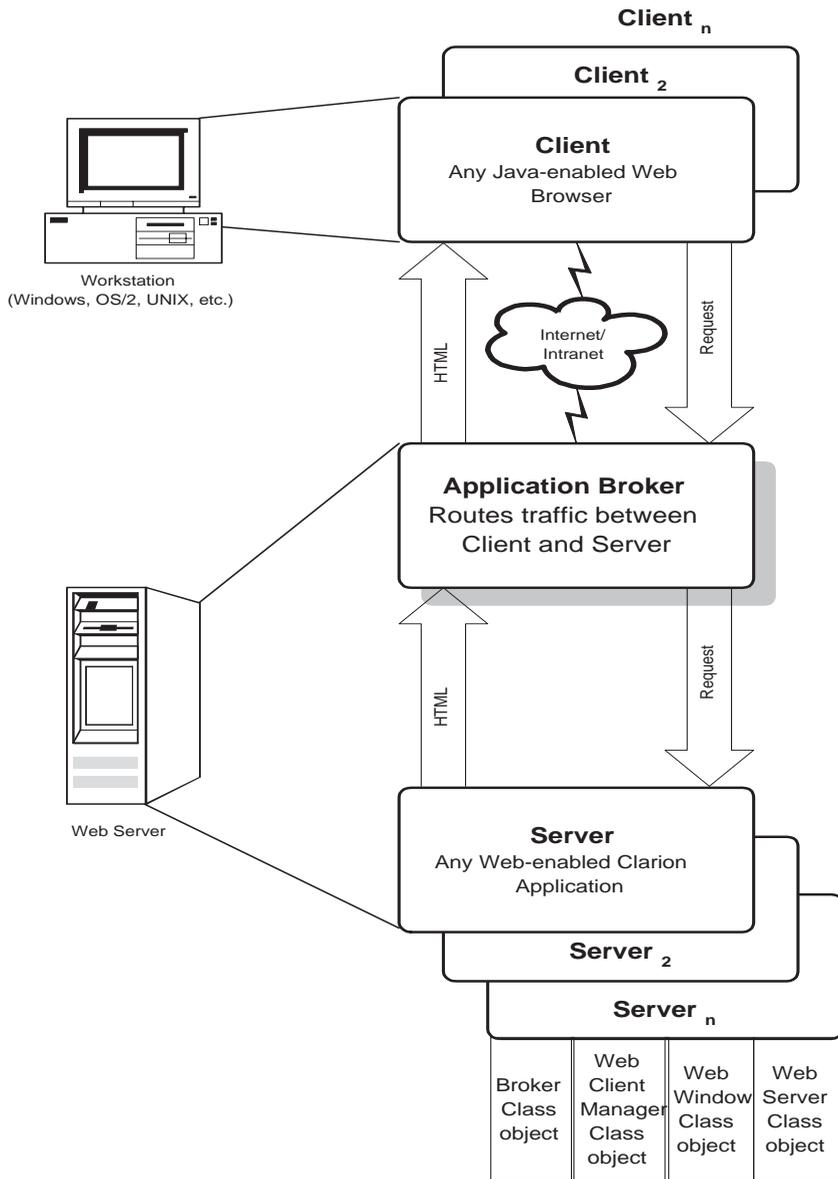
The IBC Library contains a class to represent each of these real-world objects (WebServerClass, BrokerClass, WebClientManagerClass, JSLManagerClass). This book uses WebServerClass, BrokerClass, WebClientManagerClass, JSLManagerClass, etc. to refer to discrete executable objects within the Server program.

### **Key Concepts**

Internet Connect succeeds by using the Broker (TopSpeed's Application Broker) to maintain a persistent communication between the Client (internet browser) and the Server (Web-enabled Clarion application) over an internet connection. The Client generates requests for the Server to process, and the Server generates HTML code in response to the Clients' requests. The Broker routes Client requests to the appropriate Server, and routes Server generated HTML code to the Client that requested it.

One very important point to remember is that the Server cannot initiate any action on the Client. In the Internet context, the Client initiates Server activity by sending something to the Server. The Client may send a request for more information, or it may send data entered by the end user, or both. The Server responds to the Client request, typically by generating and sending more HTML code. The generated HTML code is structured so that end user manipulation of the HTML controls sends additional requests to the Server.

The relationship between Client, Server, and Broker may be represented as shown in the following diagram.



## Internet Builder Class Synopsis

---

Following is a brief description of the functions and purposes of the main Internet Builder Classes. The Server (your Web-enabled Clarion application) program uses these classes to receive and respond to Client requests through the Broker (TopSpeed's Application Broker). The Server need not explicitly instantiate all these classes because some of the classes are instantiated by the other classes as needed.

### **BrokerClass**

The BrokerClass communicates with the Application Broker. It relies on the HttpClass to receive and process (parse) Client requests, and to forward the responses prepared by the WebWindowClass to the Client, through the Application Broker. The responses may include HTML code, data for Java applets (JSL data), graphic images (.JPG, .GIF, etc.), or any other information requested by the Client. Typically, the Server instantiates a single global BrokerClass object.

### **WebClientManagerClass**

The WebClientManagerClass helps the WebWindowClass generate appropriate HTML code and JSL data by providing accurate information about the Client's browser. The WebClientManagerClass contains current information about the Client Web browser, Java Support Library, and IP address. The WebClientManagerClass also communicates with the BrokerClass on behalf of the WebWindowClass. The BrokerClass instantiates and manages a single WebClientManagerClass object.

### **WebServerClass**

The WebServerClass establishes a communication channel with the Application Broker and handles the information coming through the channel, including information such as new page requests and value-to-field assignments forwarded from the Client's browser. The WebServerClass's primary job is to decode requests (submit strings) that come from the Client through the Application Broker. Typically, the Server instantiates a single global WebServerClass object.

### **WebWindowClass**

The WebWindowClass "translates" a Clarion WINDOW to its HTML equivalent, control by control, *and* uses the WebServerClass to process requests (submit strings) that come from the Client through the Application Broker. This class contains a variety of properties that control how the Clarion to HTML translation is accomplished. For example it contains properties to specify background, menu, toolbar, and client area colors, as well as the action to take for disabled controls, etc. Typically, the Server

instantiates a local `WebWindowClass` object for each `WINDOW` you want to display on the Client.

### **Other Classes**

There are several other classes in the IBC Library that help these main classes accomplish their objectives. All of these Internet Builder Classes are documented in the remainder of this book.

## **Internet Builder Class Header Files**

---

### **Including the right files in your program's data section**

Many of the class declarations directly reference other classes. To resolve these references, each class header (.INC file) `INCLUDES` *only* the headers containing the *directly referenced* classes. This convention maximizes encapsulation, minimizes compile times, and ensures that all necessary components are present for the make process. We recommend you follow this convention too.

A good rule of thumb is to `INCLUDE` as little as possible. The compiler will let you know if you have omitted something.

### **Including the right files in your project**

If you use the Internet Connect Templates, they automatically add the appropriate files to your project. However, if you hand code your program you must manually add some files to your project. See the *Hand Coded Programs* in the *Internet Connect User's Guide* for more information.

## **Internet Connect Templates and the IBC Library**

---

The Internet Connect Templates rely heavily on the Internet Builder Classes. However, the templates are highly configurable and are designed to let you substitute your own class definitions if you wish. See *Internet Builder Class Templates—Classes* in the *Internet Connect User's Guide* for more information on configuring the global level interaction between the Internet Connect Templates and the IBC Library. See *Internet Builder Class Templates—Classes* in the *Internet Connect User's Guide* for more information on configuring the IBC Library for individual controls.

# Documentation Conventions

## Reference Item Formats and Syntax Diagrams

---

Each Clarion programming language element referenced in this manual is printed in UPPER CASE letters.

Each Hyper-Text Markup Language (HTML) element referenced in this manual is printed in <UPPER CASE> letters surrounded by angle brackets.

Class Properties (data types and structures) occur at the beginning of a chapter, followed by Class Methods (functions and procedures) at the end. Methods are generally documented in alphabetical order rather than logical order.

The documentation formats for Properties and Methods are illustrated in the following syntax diagrams:

### Property (short description of intended use)

---

Property	Datatype, ATTRIBUTES
	A complete description of the <b>Property</b> and its uses.
	<b>Datatype</b> shows the datatype of the property such as LONG or &BrowseClass.
	<b>Attributes</b> shows any significant Clarion attributes such as PROTECTED.
Implementation:	A discussion of specific implementation issues. The implementation may change with each release / version of Internet Connect.
	<pre> ComplexDataType STRUCTURE          !actual structure declaration                                 END </pre>
See Also:	Related Methods and Properties

## Method (short description of what the method does)

```

Method( | parameter1 | [, parameter2 ] ),RETURN DATA TYPE, ATTRIBUTES
          | alternate |
          | parameter |
          | list       |

```

- Method**                      A brief statement of *what* the method does.
- parameter1*                 A complete description of parameter1, along with how it relates to parameter2 and the Method.
- parameter2*                 A complete description of parameter2, along with how it relates to parameter1 and the Method. Because it is enclosed in brackets, [ ], it is optional, and may be omitted.

A concise description of *what* the **Method** does. Sometimes a Method has no parameters.

**Attributes** shows any significant Clarion attributes such as VIRTUAL or PROTECTED.

Return Data Type:            The data type returned.

Implementation:             A description of *how* the method currently accomplishes its objective. The implementation may change with each release / version of Clarion.

Example:

```

FieldOne = FieldTwo + FieldThree            !This is a source code example
FieldThree = Method(FieldOne,FieldTwo)     !Comments follow the "!" character

```

See Also:                     Related Methods and Properties

# ***BROKER CLASS***

<b>Overview</b>	<b>41</b>
BrokerClass Concepts.....	41
Relationship to Other Internet Builder Classes .....	41
Internet Connect Template Implementation .....	42
Source Files .....	42
Conceptual Example .....	43
<b>BrokerClass Properties</b>	<b>44</b>
Client (WebClientManagerClass object) .....	44
Http (HttpClass object) .....	44
Files (WebFilesClass object) .....	45
ServerName (server identifier) .....	45
<b>BrokerClass Methods</b>	<b>46</b>
Functional Organization—Expected Use .....	46
CloseChannel (close channel to application broker) .....	47
GetAuthorizedInfo (get client password) .....	48
GetClient (return WebClientManagerClass object) .....	49
GetRequestArguments (return browser request) .....	50
Init (initialize the BrokerClass object).....	51
Kill (shut down the BrokerClass object) .....	52
OpenChannel (open channel to application broker) .....	53
ProcessHttpRequest (process incoming http) .....	54
SetClient (set client information) .....	55
SetClientBrowser (set browser information) .....	55
TakeFile (send HTML code or JSL data) .....	56
TakeHtmlPage (prepare and send HTML code) .....	57
TakeJsldata (prepare and send JSL data) .....	58
TakeUnauthorized (prepare and send access denied page) .....	59



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts* in the preceding chapter. This short topic contains information and terms that are prerequisite to the following material.

## BrokerClass Concepts

---

The BrokerClass communicates with the Application Broker. It relies on the HttpClass to receive and process (parse) Client requests, and to forward the responses prepared by the WebWindowClass to the Client, through the Application Broker. The responses may include HTML code, data for Java applets (JSL data), graphic images (.JPG, .GIF, etc.), or any other information requested by the Client. Typically, the Server instantiates a single global BrokerClass object. The Server typically instantiates a single global BrokerClass object that does the following fundamental things to aid this transmission of information:

- Records the instance number or handle of the Server
- Records the IP address of the Client that started this Server instance
- Records the directory paths that contain files shared with the Client
- Records the type of browser the Client is running
- Establishes a communication channel to the Application Broker

Most of the recorded information is supplied by the Application Broker when it starts the Server application. With this information, the Server can identify itself, its client, and the location of any shared files when it communicates with the Application Broker.

## Relationship to Other Internet Builder Classes

---

### InternetDataSinkClass

The BrokerClass is derived from the InternetDataSinkClass. The InternetDataSinkClass methods don't do anything; they are simply placeholders. The InternetDataSinkClass does provide a generic reference for all of its derived classes.

### HttpClass

The BrokerClass uses the HttpClass to process incoming and outgoing HTTP (Hyper-Text Transfer Protocol) headers. HTTP is the protocol that web

servers use to transmit HTML pages to internet browsers. Conversely, internet browsers use HTTP to transmit their requests to web servers.

### **WebClientManagerClass**

The BrokerClass uses the WebClientManagerClass to supply information about the client's browser and IP address.

### **WebFilesClass**

The BrokerClass uses the WebFilesClass to locate and identify files shared with the client.

If your program uses the BrokerClass, it also needs these other classes. See the *Conceptual Example* for more information.

## **Internet Connect Template Implementation**

---

The Internet Connect Templates generate code to instantiate a single global BrokerClass object for your web-enabled application. The BrokerClass object is called Broker. Except for the Broker initialization and termination, the template generated code does not reference the Broker object; all other references to the Broker object are handled by other IBC Library objects.

## **Source Files**

---

The BrokerClass source code is installed by default to the \LIBSRC folder. The specific BrokerClass files and their respective components are:

ICBROKER.INC	BrokerClass declarations
ICBROKER.CLW	BrokerClass method definitions

## Conceptual Example

The following example shows a typical sequence of statements to declare, instantiate, initialize, use, and terminate a BrokerClass object and related objects.

```

MyWebPgm      PROGRAM

LinkBaseClasses      EQUATE(1)
BaseClassDllMode     EQUATE(0)
  INCLUDE('ICTXTOUT.INC')           !declare TextOutputClass
  INCLUDE('ICBROKER.INC')          !declare WebBrokerClass
  INCLUDE('ICWINDOW.INC')          !declare WebWindowClass
  INCLUDE('ICCLIENT.INC')         !declare WebClientManagerClass
  INCLUDE('ICREPORT.INC')         !declare WebReportClass
  INCLUDE('ICSTD.EQU')            !declare Internet EQUATEs
MAP
  INCLUDE('ICSTD.INC')             !declare Internet helper procedures
END
!data declarations
Broker          BrokerClass      !declare Broker object
HtmlManager     HtmlClass        !declare HtmlManager object
JavaEvents      Js1EventsClass   !declare Js1Events object
WebServer       WebServerClass   !declare WebServer object
WebFileManager  WebFilesClass    !declare WebFileManager object

ICServerWin     WINDOW,AT(-100,-100,0,0) !declare an invisible window
END

CODE
WebFileManager.Init(1, '')         !initialize WebFileManager object
JavaEvents.Init                   !initialize JavaEvents object
Broker.Init('TREE', WebFileManager) !initialize Broker object
HtmlManager.Init(WebFileManager)  !initialize HtmlManager object
WebServer.Init(Broker, ,600, ,WebFileManager) !initialize WebServer object
IF (WebServer.GetInternetEnabled()) !if launched by App Broker
  OPEN(ICServerWin)               !open invisible window
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect             !set up communication w/ App Broker
    Main                          !run as usual
  BREAK
END
END
ELSE                               !if not launched by App Broker
  Main                            !run as usual
END
WebServer.Kill                    !shut down WebServer object
HtmlManager.Kill                  !shut down HtmlManager object
Broker.Kill()                    !shut down Broker object
JavaEvents.Kill                  !shut down JavaEvents object
WebFileManager.Kill              !shut down WebFileManager object

```

## ***BrokerClass Properties***

The BrokerClass contains the properties listed below.

### **Client (WebClientManagerClass object)**

---

<b>Client</b>	<b>&amp;WebClientManagerClass, PROTECTED</b>
---------------	--

The **Client** property is a reference to the WebClientManagerClass object that supplies information about the client's browser and IP address.

This property is PROTECTED, therefore, it can only be referenced by a BrokerClass method, or a method in a class derived from BrokerClass.

Implementation: The Init method calls the SetClient method to instantiate a WebClientManagerClass object for this BrokerClass object.

See Also: Init, SetClient

### **Http (HttpClass object)**

---

<b>Http</b>	<b>&amp;HttpClass</b>
-------------	-----------------------

The **Http** property is a reference to the HttpClass object that processes incoming and outgoing HTTP headers.

Implementation: The Init method instantiates an HttpClass object for this BrokerClass object.

See Also: Init

## Files (WebFilesClass object)

---

### Files    &WebFilesClass

The **Files** property is a reference to the WebFilesClass object that locates and identifies files shared with the client's browser.

Implementation:        The Init method sets the initial value of the Files property.

See Also:                Init

## ServerName (server identifier)

---

### ServerName    CSTRING(255), PROTECTED

The **ServerName** property uniquely identifies this instance of the Server (Web-enabled application). The BrokerClass object uses the ServerName property to uniquely identify the Server when communicating with the Application Broker.

This property is PROTECTED, therefore, it can only be referenced by a BrokerClass method, or a method in a class derived from BrokerClass.

Implementation:        The Init method sets the value of the ServerName property. The Application Broker supplies the ServerName value when it launches the Server program.

See Also:                Init

## BrokerClass Methods

The BrokerClass contains the methods listed below.

### Functional Organization—Expected Use

---

As an aid to understanding the BrokerClass, it is useful to organize its methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### Primary Interface Methods

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize the BrokerClass object
Kill	shut down the BrokerClass object

##### **Mainstream Use:**

##### **Occasional Use:**

CloseChannel	close channel to application broker
GetAuthorizedInfo	get client password
GetClient	return WebClientManagerClass object
GetRequestArguments	return browser request
OpenChannel	open channel to application broker
ProcessHTTPHeader	process incoming http
SetClient	set client information
SetClientBrowser	set browser information

#### Virtual Methods

Typically you will not call these methods directly—the Primary Interface methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

TakeFile	send HTML code or JSL data to client
TakeHtmlPage	prepare and send HTML code
TakeJslData	prepare and send JSL data
TakeUnauthorized	prepare and send access denied page

## CloseChannel (close channel to application broker)

---

### CloseChannel

The **CloseChannel** method closes the communication channel to the Application Broker. The communication channel is established by the **OpenChannel** method.

Example:

```
BrokerClass.Kill PROCEDURE
CODE
SELF.CloseChannel                               !close communication channel
IF (NOT (SELF.Http &= NULL))
    SELF.Http.Kill
    DISPOSE(SELF.Http)
END
IF (NOT SELF.CurClient &= NULL)
    SELF.CurClient.Kill
    DISPOSE(SELF.CurClient)
END
```

See Also: [OpenChannel](#)

## GetAuthorizedInfo (get client password)

---

**GetAuthorizedInfo**( *area, userid, password* )

**GetAuthorizedInfo** Gets the client's userid and password.

*area* A string constant, variable, EQUATE, or expression containing the name or description of the restricted Web page.

*userid* A string variable to receive the client userid.

*password* A string variable to receive the client password.

The **GetAuthorizedInfo** method gets the client's userid and password. Your program can verify the contents of the userid and password before proceeding.

When it demands a password, the client browser displays the contents of the *area* parameter to the end-user. By default, this value comes from the `WebWindowClass.AuthorizeArea` property.

Implementation: The `GetAuthorizedInfo` method relies on the `HttpClass` object to supply the userid and password.

Example:

```
WebWindowClass.GetAuthorized      PROCEDURE
Password                          STRING(255)
UserName                          STRING(255)
CODE
SELF.Server.Broker.GetAuthorizedInfo(SELF.AuthorizeArea, UserName, Password)
RETURN SELF.ValidatePassword(UserName, Password)
```

See Also: `WebWindowClass.Authorize`, `WebWindowClass.AuthorizeArea`, `WebWindowClass.GetAuthorized`, `WebWindowClass.ValidatePassword`

## GetClient (return WebClientManagerClass object)

---

### GetClient, WebClientManagerClass

The **GetClient** returns a reference to the `WebClientManagerClass` object for this `BrokerClass` object. The `WebClientManagerClass` object supplies information about the client browser.

Implementation: The `GetClient` method calls the `SetClient` method to instantiate the `BrokerClass`' `WebClientManagerClass` object.

Return Data Type: `WebClientManagerClass`

Example:

```
MyWebServerClass.Init PROCEDURE(*BrokerClass Broker,SIGNED TimeOut,WebFilesClass Files)
```

```
  ServerName          CSTRING(255)  
  StartIndex          SIGNED  
  EndIndex            SIGNED
```

```
  CODE
```

```
  SELF.CurSubmit &= NEW SubmitItemClass
```

```
  SELF.Broker &= Broker
```

```
  SELF.Client &= Broker.GetClient()
```

```
  !set reference to Client info
```

```
  !etc.
```

See Also: `SetClient`

## GetRequestArguments (return browser request)

### GetRequestArguments, STRING

The **GetRequestArguments** returns a browser request string from the Client.

**Implementation:** The **GetRequestArguments** method relies on the **HttpClass** object to supply the browser request arguments. The requests come from the HTTP Hyper-Text Transfer Protocol) the Client browser transmits to the Server (your Web-enabled application) through the Application Broker.

**Return Data Type:** **STRING**

**Example:**

```
MyWebServerClass.TakeEvent PROCEDURE

Header           EQUATE('Internet:')
CmdRequest       EQUATE('Request')
HttpStart        UNSIGNED
LenHeader        SIGNED
LenCmdRequest    SIGNED
LenCmd           SIGNED
Request          ANY
CODE
IF (SELF.Active)
CASE EVENT()
OF EVENT:DDEexecute
Request = DDEvalue()
LenHeader = LEN(Header)
LenCmdRequest = LEN(CmdRequest)
LenCmd = LenHeader + LenCmdRequest
HttpStart = INSTRING(NewLine,Request,1,LenCmd+2)+2
SELF.Broker.ProcessHttpHeader(SUB(Request,HttpStart,LEN(Request)-HttpStart))

SELF.Arguments = SELF.Broker.GetRequestArguments() !get Client browser request

SELF.Client &= SELF.Broker.GetClient()
SELF.LastRequest = CLOCK()
IF (NOT SELF.Arguments)
RETURN NET:Unknown
END
IF (SUB(SELF.Arguments,1,1) = '@')
SELF.Arguments = SUB(SELF.Arguments,2,-1)
END
IF (SELF.Arguments)
SELF.Arguments = SELF.Arguments & '&'
END
SELF.ArgIndex = 1
RETURN NET:Request
END
END
RETURN NET:Unknown
```

**See Also:** **HttpClass.GetArguments**

## Init (initialize the BrokerClass object)

**Init**( *program*, *WebFilesClass* )

<b>Init</b>	Initializes the BrokerClass object.
<i>program</i>	A string constant, variable, EQUATE, or expression containing the program name.
<i>WebFilesClass</i>	The label of the WebFilesClass object that locates and identifies files shared with the client browser.

The **Init** method initializes the BrokerClass object.

The BrokerClass uses the *program* value when communicating with the client browser, for example to identify the program for processing cookies or when demanding password authorization.

Implementation: The Init method records the client IP address, the Web-application's instance number, and calls the SetClient method to record information about the client's specific Web browser.

The Init method passes the *program* parameter to the HttpClass object to supply to the client browser as needed.

Example:

```
!data declarations
CODE
WebFilesManager.Init(1, '')           !initialize WebFilesManager object
JavaEvents.Init                      !initialize JavaEvents object
Broker.Init('TREE', WebFilesManager) !initialize Broker object
HtmlManager.Init(WebFilesManager)    !initialize HtmlManager object
WebServer.Init(Broker,,600,,WebFilesManager) !initialize WebServer object
IF (WebServer.GetInternetEnabled())  !if launched by App Broker
  OPEN(ICServerWin)                  !open invisible window
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect                !set up communication w/ App Broker
    Main                              !run as usual
    BREAK
  END
END
ELSE                                  !if not launched by App Broker
  Main                              !run as usual
END
WebServer.Kill                       !shut down WebServer object
HtmlManager.Kill                     !shut down HtmlManager object
Broker.Kill()                         !shut down Broker object
JavaEvents.Kill                      !shut down JavaEvents object
WebFilesManager.Kill                 !shut down WebFilesManager object
```

See Also: SetClient, HttpClass.SetProgName

## Kill (shut down the BrokerClass object)

### Kill

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Example:

```
!data declarations
CODE
WebFileManager.Init(1, '')           !initialize WebFileManager object
JavaEvents.Init                     !initialize JavaEvents object
Broker.Init('TREE', WebFileManager) !initialize Broker object
HtmlManager.Init(WebFileManager)    !initialize HtmlManager object
WebServer.Init(Broker,,600,,WebFileManager) !initialize WebServer object
IF (WebServer.GetInternetEnabled()) !if launched by App Broker
  OPEN(ICServerWin)                 !open invisible window
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect               !set up communication w/ App Broker
    Main                            !run as usual
    BREAK
  END
END
ELSE
  Main                              !if not launched by App Broker
  !run as usual
END
WebServer.Kill                      !shut down WebServer object
HtmlManager.Kill                    !shut down HtmlManager object
Broker.Kill()                       !shut down Broker object
JavaEvents.Kill                     !shut down JavaEvents object
WebFileManager.Kill                 !shut down WebFileManager obj
```

## OpenChannel (open channel to application broker)

---

### OpenChannel, BYTE

The **OpenChannel** method opens a communication channel to the Application Broker and returns a value indicating its success or failure. A return value of zero (0) indicates the channel is not established; any other value indicates the identity or handle of the channel.

The communication channel is dedicated to communications between the Application Broker and a single instance of the Server (Web-enabled application).

The **CloseChannel** method closes the channel established by the **OpenChannel** method.

Return Data Type: **BYTE**

Example:

```
WebServerClass.Connect          PROCEDURE
CODE
SELF.Active = SELF.Broker.OpenChannel()
IF (SELF.Active)
    IC:InitializeHooks
END
```

See Also: **CloseChannel**

## ProcessHTTPHeader (process incoming http)

### ProcessHTTPHeader( *http* )

#### ProcessHTTPHeader

Processes the Hyper-Text Transfer Protocol (HTTP) stream prepended to transmissions from the client browser.

*http*

A string constant, variable, EQUATE, or expression containing the http to process.

The **ProcessHTTPHeader** method processes the Hyper-Text Transfer Protocol (HTTP) stream prepended to transmissions from the client browser.

Implementation:

The ProcessHTTPHeader method relies on the HttpClass object to process the http stream.

Example:

```
MyWebServerClass.TakeEvent PROCEDURE
Header          EQUATE('Internet:')
CmdRequest      EQUATE('Request')
HttpStart       UNSIGNED
LenHeader       SIGNED
LenCmdRequest   SIGNED
LenCmd          SIGNED
Request         ANY
CODE
  IF (SELF.Active)
    CASE EVENT()
      OF EVENT:DDEexecute
        Request = DDEvalue()
        LenHeader = LEN(Header)
        LenCmdRequest = LEN(CmdRequest)
        LenCmd = LenHeader + LenCmdRequest
        HttpStart = INSTRING(NewLine,Request,1,LenCmd+2)+2
        SELF.Broker.ProcessHTTPHeader(SUB(Request,HttpStart,LEN(Request)-HttpStart))
        SELF.Arguments = SELF.Broker.GetRequestArguments()
        SELF.Client &= SELF.Broker.GetClient()
        SELF.LastRequest = CLOCK()
        IF (NOT SELF.Arguments)
          RETURN NET:Unknown
        END
        IF (SUB(SELF.Arguments, 1, 1) = '@')
          SELF.Arguments = SUB(SELF.Arguments,2,-1)
        END
        IF (SELF.Arguments)
          SELF.Arguments = SELF.Arguments & '&'
        END
        SELF.ArgIndex = 1
        RETURN NET:Request
      END
    END
  RETURN NET:Unknown
```

See Also:

**HttpClass.ProcessHTTPHeader**

## SetClient (set client information)

---

### SetClient

The **SetClient** method records information about the Client.

Implementation: The **SetClient** method instantiates a **WebClientManagerClass** object if necessary, then calls the **SetClientBrowser** method to identify the Client Web browser.

Example:

```
BrokerClass.Init PROCEDURE (STRING ProgramName, WebFilesClass Files)
  CODE
  IE30.Init (BROWSER:IE30, TRUE, FALSE)
  IE40.Init (BROWSER:IE40, TRUE, FALSE)
  NetScape3x.Init (BROWSER:NetScape3, FALSE, TRUE)
  NetScape3x.SetNoCache = TRUE
  Mozilla4.Init (BROWSER:Mozilla4, FALSE, TRUE)
  Mozilla4.SetNoCache = TRUE
  UnknownBrowser.Init (BROWSER:Unknown, FALSE, FALSE)
  SELF.ServerName = IC:GetCommandLineOption('/inet=')
  SELF.Files &= Files
  SELF.Http &= NEW HttpClass
  SELF.Http.Init (Files)
  SELF.Http.SetProgName (ProgramName)
  SELF.SetClient !set Client information
  SELF.CurClient.IP = IC:GetCommandLineOption('/client=')
```

See Also: [SetClientBrowser](#)

## SetClientBrowser (set browser information)

---

### SetClientBrowser, PROTECTED

The **SetClientBrowser** method records information about the Client's Web browser.

This method is **PROTECTED**, therefore, it can only be called by a **BrokerClass** method, or a method in a class derived from **BrokerClass**.

Implementation: The **SetClientBrowser** method notes the Client's specific type of Web browser (Netscape, Internet Explorer, etc.), its version number, and the facilities it supports, such as caching, java, style sheets, etc.

Example:

```
BrokerClass.SetClient PROCEDURE
  CODE
  IF (SELF.CurClient &= NULL)
    SELF.CurClient &= NEW WebClientManagerClass
    SELF.CurClient.Init (SELF, UnknownBrowser)
  END
  SELF.SetClientBrowser
```

## TakeFile (send HTML code or JSL data)

**TakeFile( *filename, secure, immediate* ), VIRTUAL**

<b>TakeFile</b>	Requests the Application Broker to transmit HTML code or JSL data to the Client browser.
<i>filename</i>	A string constant, variable, EQUATE, or expression naming the file containing the HTML code or JSL data to transmit.
<i>secure</i>	An integer constant, variable, EQUATE, or expression indicating whether to transmit over a secure channel or public channel. A value of Secure:Full indicates a secure channel; a value of Secure:None indicates a public channel; and a value of Secure:Default indicates the default channel.
<i>immediate</i>	An integer constant, variable, EQUATE, or expression indicating whether to immediately transmit the information or to hold the transmission until the browser requests the information. A value of one (1 or True) does an immediate transmit, which is appropriate for JSL data. A value of zero (0 or False) forces the browser to make a subsequent request for a new page, which is appropriate for HTML code.

The **TakeFile** method requests the Application Broker to transmit HTML code or JSL data to the Client browser.

TakeFile is a VIRTUAL method so that other base class methods can directly call the TakeFile virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: EQUATEs for the *secure* parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

Example:

```

BrokerClass.TakeJslData PROCEDURE(STRING Filename, SIGNED Security)

CODE
SELF.Http.PreparePageForJava(200, Filename)
SELF.Http.FinishPage()
SELF.TakeFile(Filename, Security, TRUE)

```

## TakeHtmlPage (prepare and send HTML code)

**TakeHtmlPage( *filename*, *secure*, *immediate* ), VIRTUAL**

<b>TakeHtmlPage</b>	Prepares an HTML page then requests the Application Broker to transmit the HTML to the Client browser.
<i>filename</i>	A string constant, variable, EQUATE, or expression naming the file containing the HTML code to transmit.
<i>secure</i>	An integer constant, variable, EQUATE, or expression indicating whether to transmit over a secure channel or public channel. A value of Secure:Full indicates a secure channel; a value of Secure:None indicates a public channel; and a value of Secure:Default indicates the default channel.
<i>immediate</i>	An integer constant, variable, EQUATE, or expression indicating whether to immediately transmit the information or to hold the transmission until the browser requests the information. A value of one (1 or True) does an immediate transmit, which is appropriate for JSL data. A value of zero (0 or False) forces the browser to make a subsequent request for a new page, which is appropriate for HTML code.

The **TakeHtmlPage** method prepares an HTML page then requests the Application Broker to transmit the HTML to the client browser.

TakeHtmlPage is a VIRTUAL method so that other base class methods can directly call the TakeHtmlPage virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

TakeHtmlPage uses the HttpClass object to prepend the appropriate HTTP to the HTML, then calls the TakeFile method to transmit the whole package.

EQUATEs for the *secure* parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

Example:

```

WebClientManagerClass.TakeHtmlPage PROCEDURE |
    (STRING Filename,SIGNED Security, BYTE dontmove=FALSE)
CODE
SELF.Broker.TakeHtmlPage(Filename, Security, dontmove)

```

See Also:

TakeFile

## TakeJsldata (prepare and send JSL data)

### TakeJsldata( *filename*, *secure* ), VIRTUAL

<b>TakeJsldata</b>	Prepares Java Support Library (JSL) data then requests the Application Broker to transmit the JSL data to the client browser.
<i>filename</i>	A string constant, variable, EQUATE, or expression naming the file containing the JSL data to transmit.
<i>secure</i>	An integer constant, variable, EQUATE, or expression indicating whether to transmit over a secure channel or public channel. A value of Secure:Full indicates a secure channel; a value of Secure:None indicates a public channel; and a value of Secure:Default indicates the default channel.

The **TakeJsldata** method prepares Java Support Library (JSL) data then requests the Application Broker to transmit the JSL data to the client browser.

TakeJsldata is a VIRTUAL method so that other base class methods can directly call the TakeJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

TakeJsldata uses the HttpClass object to prepend the appropriate HTTP to the JSL data, then calls the TakeFile method to transmit the whole package.

EQUATEs for the *secure* parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

Example:

```

WebClientManagerClass.TakeJsldata PROCEDURE(STRING Filename,SIGNED Security)
CODE
SELF.Broker.TakeJsldata(Filename, Security)

```

See Also:

TakeFile

## TakeUnauthorized (prepare and send access denied page)

**TakeUnauthorized( *filename*, *secure* ), VIRTUAL**

**TakeUnauthorized** Prepares an “access denied” page then requests the Application Broker to transmit the page to the client browser.

*filename* A string constant, variable, EQUATE, or expression naming the file containing the “access denied” page to transmit.

*secure* An integer constant, variable, EQUATE, or expression indicating whether to transmit over a secure channel or public channel. A value of Secure:Full indicates a secure channel; a value of Secure:None indicates a public channel; and a value of Secure:Default indicates the default channel.

The **TakeUnauthorized** method prepares an “access denied” page then requests the Application Broker to transmit the page to the client browser.

TakeUnauthorized is a VIRTUAL method so that other base class methods can directly call the TakeUnauthorized virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

TakeUnauthorized uses the HttpClass object to prepend the appropriate HTTP to the “access denied” page, then calls the TakeFile method to transmit the whole package.

EQUATEs for the *secure* parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

Example:

```

WebClientManagerClass.TakeUnauthorized PROCEDURE(STRING Filename,SIGNED Security)
CODE
SELF.Broker.TakeUnauthorized(Filename, Security)

```

See Also:

TakeFile



# WEB SERVER CLASSES

<b>Overview</b>	<b>63</b>
WebServerClass Concepts .....	63
ShutDownClass Concepts .....	63
Relationship to Other Internet Builder Classes .....	63
Internet Connect Template Implementation .....	64
Source Files .....	64
Conceptual Example .....	65
<b>WebServerClass Properties</b>	<b>66</b>
Active (server communicates with broker) .....	66
ArgIndex (index to browser request string) .....	66
Broker (BrokerClass object) .....	66
CurSubmit (SubmitItemClass object) .....	67
Client (WebClientManagerClass object) .....	67
CommandLine (command line parameters) .....	68
DialogPageBackColor (MESSAGE page background color) .....	68
DialogPageImage (MESSAGE page wallpaper) .....	68
DialogWinBackColor (MESSAGE window background color) .....	68
DialogWinImage (MESSAGE window wallpaper) .....	69
Files (WebFilesClass object) .....	69
GotCommandLine (command line arguments set flag) .....	69
JavaLibraryPath (Java Support Library location) .....	70
PageToReturnTo (return URL) .....	70
ProgramName (Server pathname) .....	71
TimeOut (period of inactivity after which to shut down) .....	71
<b>WebServerClass Methods</b>	<b>72</b>
Functional Organization—Expected Use .....	72
Connect (establish communication with Application Broker) .....	74
GetInternetEnabled (return Web/Windows mode) .....	75
GetReadyForPage (return ready-to-continue flag) .....	76
GetRequestedWholePage (return scope of browser request) .....	77
GetSendWholePage (return full or partial refresh flag) .....	78
Halt (immediately shut down the Server) .....	79
Init (initialize the WebServerClass object) .....	80
Kill (shut down the WebServerClass object) .....	81

Quit (shut down the Server normally) ..... 82

SetDialogPageBackground (set MESSAGE Web page background)..... 83

SetDialogWinBackground (set MESSAGE Web page window background) ..... 84

SetSendWholePage (force full page refresh) ..... 85

SetNewPageDisable (suppress outgoing Web pages) ..... 86

SetNextAction (return Web page field information) ..... 87

TakeEvent (process WebServerClass object events) ..... 88

TakePageSent (prepare WebServerClass object for next page) ..... 89

**ShutDownClass Methods** **90**

Close (a virtual shut down the Web-enabled application) ..... 90

## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### WebServerClass Concepts

---

The WebServerClass establishes a communication channel with the Application Broker and handles the information coming through the channel, including information such as new page requests and value-to-field assignments forwarded from the Client's browser. The WebServerClass's primary job is to decode requests (submit strings) that come from the Client through the Application Broker. Typically, the Server instantiates a single global WebServerClass object.

### ShutDownClass Concepts

---

The WebServerClass uses the ShutDownClass to shut down other IBC Library objects when the Server (Web-enabled application) undergoes a normal shut down, including end user request and time outs.

### Relationship to Other Internet Builder Classes

---

The WebServerClass uses the SubmitItemClass to do much of its work. That is, the WebServerClass uses the SubmitItemClass to process information from the Client's browser.

The WebServerClass uses the ShutDownClass to shut down other IBC Library objects when the Server (Web-enabled application) undergoes a normal shut down, including end user request and time outs.

The WebServerClass also references the BrokerClass, the WebFilesClass, and the WebClientManagerClass. Therefore, if your program uses the WebServerClass, it also needs these other classes. See the *Conceptual Example* for more information.

The WebWindowClass and the WebReportClass rely on the WebServerClass to communicate with the Application Broker, to supply various pieces of information (files and process status), and to process events related to program timeouts and Application Broker communications.

## Internet Connect Template Implementation

---

The Internet Connect Templates generate code to instantiate a single global WebServerClass object for your application program. The WebServerClass object is called WebServer. The templates generate code to initialize the WebServer object, to establish the communication channel with the Application Broker, and to shut down the WebServer object when the program ends. All other references to the WebServer object are made in the static IBC Library code and not within the template generated code.

## Source Files

---

The WebServerClass source code is installed by default to the \LIBSRC folder. The specific WebServerClass files and their respective components are:

ICSERVER.INC	WebServerClass declarations ShutDownClass declarations
ICSERVER.CLW	WebServerClass method definitions ShutDownClass method definitions
ICSERVER.TRN	WebServerClass translation strings

## Conceptual Example

The following example shows a typical sequence of statements to declare, instantiate, initialize, use, and terminate a `WebServerClass` object and related objects.

```

MyWebPgm      PROGRAM

LinkBaseClasses      EQUATE(1)
BaseClassDllMode     EQUATE(0)
  INCLUDE('ICTXTOUT.INC')           !declare TextOutputClass
  INCLUDE('ICBROKER.INC')           !declare WebBrokerClass
  INCLUDE('ICWINDOW.INC')           !declare WebWindowClass
  INCLUDE('ICCLIENT.INC')           !declare WebClientManagerClass
  INCLUDE('ICSTD.EQU')               !declare Internet EQUATES
MAP
  INCLUDE('ICSTD.INC')               !declare Internet helper procedures
END
!data declarations
Broker           BrokerClass           !declare Broker object
HtmlManager      HtmlClass             !declare HtmlManager object
JavaEvents       Js1EventsClass        !declare Js1Events object
WebServer        WebServerClass        !declare WebServer object
WebFileManager   WebFilesClass         !declare WebFileManager object
ShutDownManager CLASS(ShutDownClass)  !declare ShutDownManager object
Close            PROCEDURE,VIRTUAL
                END

ICServerWin      WINDOW,AT(-100,-100,0,0) !declare an invisible window
                END

CODE
WebFileManager.Init(1, '')           !initialize WebFileManager object
JavaEvents.Init
Broker.Init('TREE', WebFileManager) !initialize Broker object
HtmlManager.Init(WebFileManager)     !initialize HtmlManager object
WebServer.Init(Broker,ShutDownManager,,600,,WebFileManager)!init WebServer object
IF (WebServer.GetInternetEnabled())  !if launched by App Broker
  OPEN(ICServerWin)                  !open invisible window
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect                 !set up communication w/ App Broker
    Main                               !run as usual
  BREAK
END
END
ELSE
  Main                               !if not launched by App Broker
  !run as usual
END
ShutDownManager.Close                !shut down IBC Library objects

ShutDownManager.Close
CODE
WebServer.Kill                       !shut down WebServer object
HtmlManager.Kill                     !shut down HtmlManager object
Broker.Kill()                         !shut down Broker object
JavaEvents.Kill                      !shut down JavaEvents object
WebFileManager.Kill                   !shut down WebFileManager object

```

## WebServerClass Properties

### Active (server communicates with broker)

---

#### Active SIGNED(False)

The **Active** property indicates whether a communication channel is established between the Server (your Web-enabled application) and the Application Broker that launched it. A value of zero (False) indicates the channel is not established and the Server cannot be run from a Web browser. Any other value indicates the channel is established and the Server can be run from a Web browser.

Implementation: The Connect method sets the Active property to a non-zero value when it establishes a communication channel with the Application Broker.

See Also: Connect

### ArgIndex (index to browser request string)

---

#### ArgIndex SIGNED

The **ArgIndex** property indexes start and end positions of the discrete sections of the request string forwarded to the server from the client browser by the Application Broker.

Implementation: The SetNext Action method uses the ArgIndex property to parse browser requests.

See Also: SetNextAction

### Broker (BrokerClass object)

---

#### Broker &BrokerClass

The **Broker** property is a reference to a BrokerClass object. The WebServerClass object uses this property to handle all communications with the Application Broker.

Implementation: The Broker property handles the transmission of HTML code and Java Support Library (JSL) data from the Server (your Web-enabled application program) to the Application Broker.

## CurSubmit (SubmitItemClass object)

---

### CurSubmit      &SubmitItemClass, PROTECTED

The **CurSubmit** property is a reference to a SubmitItemClass object. The WebServerClass object uses this property to manage incoming data from the client browser.

This property is PROTECTED, therefore, it can only be referenced by a WebServerClass method, or a method in a class derived from WebServerClass.

Implementation:

When the end user enters data through the Web browser, the browser submits the data in a “request string” which the Application Broker forwards to the Server (your Web-enabled application). The request string optionally contains a list of field assignments of the form ‘field=value&field2=value2’. The CurSubmit property represents a single field assignment within the request string, including the field equate of the control, the new value entered in the browser, and any event associated with the control (accepted, scroll, etc.).

The Init method sets the initial value of the CurSubmit property to reference a new SubmitItemClass object. The SetNextAction method “positions” the CurSubmit property to the next field assignment.

See Also:

Init, SetNextAction

## Client (WebClientManagerClass object)

---

### Client      &WebClientManagerClass

The **Client** property is a reference to the WebClientManagerClass object that supplies information about the client’s browser and IP address. The WebServerClass object does not use the Client property, except to make it available to other objects, such as the WebWindowClass object, that reference the WebServerClass object.

Implementation:

The Init method sets the initial value of the Client property. The TakeEvent method refreshes the Client property in case the end user modifies the browser capabilities in mid-session.

See Also:

Init, TakeEvent

## CommandLine (command line parameters)

---

### CommandLine ANY

The **CommandLine** property contains any parameter string specified when the client browser requests the Server (your Web-enabled application). The `WebServerClass` object does not use this property, except to make it available to your program for parsing and processing as needed.

Implementation: The `Init` method sets the value of the `CommandLine` property. See *Using Command Line Parameters* for more information on specifying parameters.

See Also: `Init`

## DialogPageBackColor (MESSAGE page background color)

---

### DialogPageBackColor LONG

The **DialogPageBackColor** property indicates the background color of the Web page which displays Clarion MESSAGES.

Implementation: The `SetDialogPageBackground` method sets the value of the `DialogPageBackColor` property.

See Also: `SetDialogPageBackground`

## DialogPageImage (MESSAGE page wallpaper)

---

### DialogPageImage CSTRING(FILE:MaxFileName)

The **DialogPageImage** property indicates the background image (wallpaper) of the Web page which displays Clarion MESSAGES.

Implementation: The `SetDialogPageBackground` method sets the value of the `DialogPageBackColor` property.

See Also: `SetDialogPageBackground`

## DialogWinBackColor (MESSAGE window background color)

---

### DialogWinBackColor LONG

The **DialogWinBackColor** property indicates the background color of the Web page window which displays Clarion MESSAGES.

Implementation: The `SetDialogWindowBackground` method sets the value of the `DialogWinBackColor` property.

See Also: `SetDialogWindowBackground`

## DialogWinImage (MESSAGE window wallpaper)

---

DialogWinImage	CSTRING(FILE:MaxFileName)
----------------	---------------------------

The **DialogWinImage** property indicates the background image (wallpaper) of the Web page window which displays Clarion MESSAGES.

Implementation: The SetDialogWindowBackground method sets the value of the DialogWinImage property.

See Also: SetDialogWindowBackground

## Files (WebFilesClass object)

---

Files	&WebFilesClass
-------	----------------

The **Files** property is a reference to the WebFilesClass object that locates and identifies files shared with the client's browser.

Implementation: The Init method sets the initial value of the Files property.

See Also: Init

## GotCommandLine (command line arguments set flag)

---

GotCommandLine	BYTE
----------------	------

The **GotCommandLine** property indicates whether the WebServerClass has already collected command line arguments from the first browser request and stored them in the CommandLine property. A value of one (1 or True) indicates the command line parameters are collected and stored. A value of zero (0 or False) indicates the command line parameters are not yet collected and stored.

Implementation: The Init method sets the GotCommandLine property to zero. The TakeRequest method sets GotCommandLine to true after it processes the first browser request.

See Also: CommandLine, Init, TakeRequest

## JavaLibraryPath (Java Support Library location)

---

### JavaLibraryPath CSTRING(FILE:MaxFileName)

The **JavaLibraryPath** property contains the non-default location of the Java Support Library. This may be a pathname relative to the Application Broker, or it may be a URL naming another site.

The WebServerClass object does not use the JavaLibraryPath property, except to make it available to other objects, such as the WebWindowClass object, that reference the WebServerClass object.

Implementation:

The Init method sets the initial value of the JavaLibraryPath property. This value comes from the Internet Application Extension template's Advanced tab. If the Java Support Library is installed to its default location (the /public directory below the Application Broker directory), then JavaLibraryPath is null.

See Also:

Init

## PageToReturnTo (return URL)

---

### PageToReturnToCSTRING(FILE:MaxFileName)

The **PageToReturnTo** property contains the destination to go to when the Server (your Web-enabled application) shuts down normally. This may be a pathname relative to the Application Broker, or it may be a URL naming another site.

Implementation:

The Init method sets the initial value of the PageToReturnTo property. This value comes from the Internet Application Extension template's Advanced tab. If the Java Support Library is installed to its default location (the /public directory below the Application Broker directory), then JavaLibraryPath is null.

See Also:

Init

## ProgramName (Server pathname)

---

**ProgramName** CSTRING(FILE:MaxFileName)

The **ProgramName** property contains the pathname of the Server (your Web-enabled application) relative to the Application Broker. The WebServerClass object uses this property to determine whether the program was launched by the Application Broker at the request of a Web browser.

Implementation: The Init method sets the initial value of the ProgramName property. This value comes from the Application Broker when it launches the Server.

See Also: Init

## TimeOut (period of inactivity after which to shut down)

---

**TimeOut** SIGNED

The **TimeOut** property contains the number of seconds of inactivity, after which the Server (your Web-enabled application) automatically shuts down.

Implementation: The Init method sets the initial value of the TimeOut property. This value comes from the Internet Application Extension template's Advanced tab.

See Also: Init

## WebServerClass Methods

The WebServerClass inherits all the methods of the WebDataSinkClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebServerClass contains the methods listed below.

### Functional Organization—Expected Use

---

As an aid to understanding the WebServerClass, it is useful to organize the various WebServerClass methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### Primary Interface Methods

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize the WebServerClass object
SetDialogPageBackground	set MESSAGE Web page background
SetDialogWinBackgroundset	MESSAGE Web window background
GetInternetEnabled	return Web/Windows mode
Connect	establish communication to App Broker
Kill	shut down the WebServerClass object

##### **Mainstream Use:**

-none-

##### **Occasional Use:**

GetReadyForPage	return ready-to-continue flag
GetRequestedWholePage	return scope of browser request
GetSendWholePage	return full or partial refresh flag
Halt	shut down the Server immediately
Quit <sup>v</sup>	shut down the Server normally
SetSendWholePage	force full page refresh
SetNewPageDisable	suppress outgoing Web pages
SetNextAction	return Web page field information
TakeEvent	process WebServerClass object events
TakePageSent	prepare WebServerClass object for page

<sup>v</sup> These methods are also Virtual.

## **Virtual Methods**

Typically you will not call these methods directly—other IBC Library methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

Quit

shut down the Server

## Connect (establish communication with Application Broker)

---

### Connect

The **Connect** method establishes a communication channel between this instance of the Server (your Web-enabled application) and the Application Broker that launched it.

Implementation: The Connect method sets the Active property to a non-zero value when it successfully establishes the communication channel.

Example:

```
WebWindow PROGRAM
!data declarations
CODE
WebServer.Init(Broker, '', 600, '', WebFileManager)
IF (WebServer.GetInternetEnabled())
  OPEN(ICServerWin)
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect           !set up channel to App Broker
    Main
    BREAK
  END
END
ELSE
  Main
END
WebServer.Kill
```

See Also: [Active](#)

## GetInternetEnabled (return Web/Windows mode)

---

### GetInternetEnabled, BYTE

The **GetInternetEnabled** method returns a value indicating whether the Server (your Web-enabled application) was launched by the Application Broker at the request of a Web browser. A value of one (1) indicates the Server was launched by the Application Broker (Web mode); a value of zero (0) indicates the Server was not launched by the Application Broker (Windows mode).

Implementation: The GetInternetEnabled method evaluates the ProgramName property to determine whether the Server was launched by the Application Broker.

Return Data Type: **BYTE**

Example:

```
WebWindow PROGRAM
!data declarations
CODE
WebServer.Init(Broker, '', 600, '', WebFileManager)
IF (WebServer.GetInternetEnabled())           !launched from browser?
  OPEN(ICServerWin)
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect
    Main
    BREAK
  END
END
ELSE                                           !not launched from browser?
  Main
END
WebServer.Kill
```

See Also: **ProgramName**

## GetReadyForPage (return ready-to-continue flag)

### GetReadyForPage, BYTE

The **GetReadyForPage** method returns a value indicating whether the Server (your Web-enabled application) is ready to refresh the client's Web browser display. A return value of one (1) indicates the Server is ready to proceed; a value of zero (0) indicates the Server is not ready to proceed.

You can use the **SetNewPageDisable** method to instruct the **GetReadyForPage** method to return a value of zero (0). This lets you delay HTML generation based on any events or conditions you choose.

**Implementation:** The **GetReadyForPage** method returns a value of zero (0) primarily if there are events remaining to be processed. This allows the Internet Builder Class objects to delay the HTML generation until all normal Windows processing is complete.

**Return Data Type:** BYTE

**Example:**

```

WebWindowClass.TakeEvent  PROCEDURE

CODE
IF (SELF.Server.Active)
CASE SELF.Server.TakeEvent()
OF NET:Request
SELF.TakeRequest
END
CASE (EVENT())
OF EVENT:NewPage
IF (SELF.Server.GetReadyForPage())      !ready to refresh the web page?
SELF.TakeCreatePage
END
OF EVENT:OpenWindow
SELF.ResetFromControls
END
WebActiveFrame.TakeEvent
IF (SELF.Server.GetReadyForPage())      !ready to refresh the web page?
POST(EVENT:NewPage)
END
END
END

```

**See Also:** SetNewPageDisable

## GetRequestedWholePage (return scope of browser request)

---

### GetRequestedWholePage, BYTE

The **GetRequestedWholePage** method returns a value indicating whether the Client browser requested a full page refresh or a partial page refresh. A return value of one (1) indicates the Client requested a full page (slower); a value of zero (0) indicates the Client requested a partial page (faster).

Return Data Type:      **BYTE**

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
!data declarations
Index  SIGNED,AUTO
CODE
  IF SELF.Server.GetRequestedWholePage()           !full page requested?
    !clear all check boxes before sending new page
  LOOP Index = 1 TO RECORDS(SELF.Controls)
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()
  END
END
LOOP
!etc
END
```

## GetSendWholePage (return full or partial refresh flag)

### GetSendWholePage, BYTE

The **GetSendWholePage** method returns a value indicating whether the Server (your Web-enabled application) will honor or ignore the Client's (browser) request for a partial page. A return value of one (1) indicates the Server will ignore the partial page request and send a full page; a value of zero (0) indicates the Server will send whatever the Client requested.

You can use the **SetSendWholePage** method to determine the value **GetSendWholePage** returns. This lets you force a full page refresh based on any events or conditions you choose.

Return Data Type:     **BYTE**

Example:

```
MyWebWindowClass.TakeCreatePage     PROCEDURE

Client         &WebClientManagerClass
Filename       CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client

SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage())             !full page refresh?
    Filename = SELF.Files.GetFilename(Content:Html)
    IF (SELF.Server.GetRequestedWholePage())
        Client.NextHtmlPage
        SELF.CreateHtmlPage(SELF.HtmlTarget,Filename)
    ELSE
        SELF.CreateDummyHtmlPage(SELF.HtmlTarget,Filename)
    END
    Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(),FALSE)
ELSE                                             !or partial page refresh?
    Client.Js1.OpenChannel(SELF.GetTargetSecurity(),SELF.Files)
    SELF.CreateJs1Data(Client.Js1)
    Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()
```

See Also:             **SetSendWholePage**

## Halt (immediately shut down the Server)

---

### Halt

The **Halt** method shuts down the Server (your Web-enabled application) immediately. It does not attempt to preserve pending transactions or offer the end user an opportunity to continue the session.

Implementation:

The Halt method calls the `ShutdownClass.Close` method to shutdown the Server.

You can call the Halt method to let the end user exit the Server from any of its generated Web pages, rather than requiring the end user to navigate to the Server's opening page before exiting.

See Also:

`ShutdownClass.Close`

## Init (initialize the WebServerClass object)

**Init**( *broker*, *shut down* [,*return page*], *timeout* [,*JSL path*], *files* )

<b>Init</b>	Initializes the WebServerClass object.
<i>broker</i>	The label of the BrokerClass object that handles the transmission of HTML code and Java Support Library (JSL) data from the Server (your Web-enabled application program) to the Application Broker.
<i>shut down</i>	The label of the ShutDownClass object that handles the Server's normal shut down.
<i>return page</i>	A string constant, variable, EQUATE, or expression containing the destination to go to when the Server shuts down normally. If omitted, the Server supplies a standard termination page indicating it is no longer running.
<i>timeout</i>	An integer constant, variable, EQUATE, or expression containing the number of seconds of inactivity after which the Server automatically shuts down.
<i>JSL path</i>	A string constant, variable, EQUATE, or expression containing the non-default location of the Java Support Library. If omitted, the Server looks for the Java Support Library in its local directories.
<i>files</i>	The label of the FilesClass object that manages the pathname information (Java Support Library files, temporary and permanent files, directories, aliases, public and secure channels) for the Server.

The **Init** method initializes the WebServerClass object.

Implementation: The Init method sets the initial value of the Broker, Files, JavaLibraryPath, PageToReturnTo, and TimeOut properties.

Example:

```
WebWindow PROGRAM
!data declarations
CODE
WebServer.Init(Broker,ShutDown,,600,','WebFilesManager) !initialize WebServer object
OPEN(ICServerWin)
ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect
    Main
    BREAK
  END
END
ShutDown.Close !shut down Server
```

See Also: Broker, Files, JavaLibraryPath, PageToReturnTo, TimeOut

## Kill (shut down the WebServerClass object)

---

### Kill

The **Kill** method frees all memory allocated during the life of the object and performs any other required termination code.

Implementation:

The **Kill** method removes all temporary files created during the remote computing session, sends the browser to the appropriate return page, and closes the communication channel with the Application Broker.

Example:

```
WebWindow PROGRAM
!data declarations
CODE
WebServer.Init(Broker,ShutDown,,600,','WebFileManager) !initialize WebServer object
OPEN(ICServerWin)
ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect
    Main
    BREAK
  END
END
ShutDown.Close !shut down Server

ShutDown.Close PROCEDURE
CODE
WebServer.Kill !shut down WebServer object
```

## Quit (shut down the Server normally)

### Quit, VIRTUAL

The **Quit** method initiates normal shut down of the Server (your Web-enabled application). This allows executing procedures to attempt to preserve pending transactions or offer the end user an opportunity to continue the session.

Quit is a VIRTUAL method so that other base class methods can directly call the Quit virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The Quit method sets a flag to begin normal program shut down.

You can call the Quit method to let the end user exit the Server from any of its generated Web pages, rather than requiring the end user to navigate to the Server's opening page before exiting.

Example:

```

WebServerClass.TakeEvent          PROCEDURE
!data declarations

CODE
IF (SELF.Active)
  IF SELF.Aborting THEN RETURN Net:Terminate.

CASE EVENT()
OF EVENT:Terminate                !on end user request
  SELF.Quit                       ! initiate normal shut down
  RETURN Net:Terminate
OF EVENT:Request
  IF SELF.RequestPending
    SELF.RequestPending = FALSE
    RETURN SELF.TakeRequest(IC:GetRequestText())
  END
OF EVENT:Timer
  IF (SELF.LastRequest)
    IF (CLOCK()-SELF.LastRequest>SELF.TimeOut*100) !on program time out
      SELF.Quit                                     ! initiate normal shut down
    END
  END
OF EVENT:OpenWindow
  IF SELF.RequestPending
    POST(EVENT:Request)
  END
  IF O{PROP:timer}=0
    O{PROP:timer} = SELF.TimeOut * 10
  END
END
END
RETURN NET:Unknown

```

See Also: **Kill**

## SetDialogPageBackground (set MESSAGE Web page background)

**SetDialogPageBackground( [color] [,image] )**

### SetDialogPageBackground

Sets the MESSAGE Web page background color and image.

*color* An integer constant, variable, EQUATE, or expression indicating the PAGE color.

*image* A string constant, variable, EQUATE, or expression containing a filename. The WebWindowClass object displays (tiles) the image in the specified file as the background to the Web page.

The **SetDialogPageBackground** method sets the background color and image for the Web page that displays Clarion MESSAGES.

Implementation: The SetDialogPageBackground method sets the DialogPageBackColor property and the DialogPageImage properties.

Example:

```
MyWebPgm      PROGRAM
!data declarations
CODE
WebFileManager.Init(1, '')
JavaEvents.Init
Broker.Init('TREE', WebFileManager
HtmlManager.Init(WebFileManager)
WebServer.Init(Broker, ShutDownManager, ,600, ,WebFileManager)
WebServer.SetDialogPageBackground(, 'MyLogo.Gif') !set global MESSAGES logo
WebServer.SetDialogWindowBackground(COLOR:Blue) !set global MESSAGES window color
!program code
```

See Also: DialogPageBackColor, DialogPageImage

## SetDialogWinBackground (set MESSAGE Web page window background)

**SetDialogWinBackground**( [*color*] [,*image*] )

### SetDialogWinBackground

Sets the MESSAGE Web page window background color and image.

*color* An integer constant, variable, EQUATE, or expression indicating the PAGE color.

*image* A string constant, variable, EQUATE, or expression containing a filename. The WebWindowClass object displays (tiles) the image in the specified file as the background to the Web page window.

The **SetDialogWinBackground** method sets the background color and image for the Web page window that displays Clarion MESSAGES.

Implementation: The SetDialogWinBackground method sets the DialogWinBackColor property and the DialogWinImage properties.

Example:

```
MyWebPgm      PROGRAM
!data declarations
CODE
WebFileManager.Init(1, '')
JavaEvents.Init
Broker.Init('TREE', WebFileManager
HtmlManager.Init(WebFileManager)
WebServer.Init(Broker,ShutdownManager,,600,,WebFileManager)
WebServer.SetDialogPageBackground('MyLogo.Gif')!set global MESSAGES logo
WebServer.SetDialogWindowBackground(COLOR:Blue) !set global MESSAGES window color
!program code
```

See Also: DialogWinBackColor, DialogWinImage

## SetSendWholePage (force full page refresh)

---

### SetSendWholePage( *force* )

#### SetSendWholePage

Determines whether the Server honors or ignores Client (browser) requests for a partial page update.

*force* An integer constant, variable, EQUATE, or expression indicating whether the Server (your Web-enabled application) sends a full page refresh, regardless of whether the Client browser requested a full page or a partial page. A value of one (1) forces the Server to send a full page; a value of zero (0) lets the Server send whatever the Client requested.

The **SetSendWholePage** method determines whether the Server honors Client (browser) requests for a partial page update by sending only a partial page, or ignores the partial page request and sends a full page instead.

You can use the **SetSendWholePage** method to force a full page refresh based on any events or conditions you choose. Conversely, you can use the **SetSendWholePage** method to return control to the Client based on any events or conditions you choose.

Example:

```
MyWebWindowClass.Init PROCEDURE(*WebServerClass Server,*HtmlClass HtmlTarget)
```

```
CODE
```

```
! initialize WebWindow object
```

```
SELF.Server.SetSendWholePage(True)           !Always force full page on a new window
```

See Also:

**GetSendWholePage**

## SetNewPageDisable (suppress outgoing Web pages)

---

### SetNewPageDisable( *ignore* )

#### SetNewPageDisable

Determines whether the Server (your Web-enabled application) honors or ignores Client (browser) requests for information.

#### *ignore*

An integer constant, variable, EQUATE, or expression indicating whether the Server honors or ignores Client (browser) requests for information. A value of one (1) ignores Client requests for information; a value of zero (0) honors Client requests.

The **SetNewPageDisable** method determines whether the Server honors or ignores Client (browser) requests for information. This lets you delay HTML generation based on any events or conditions you choose.

An *ignore* value of one (1) tells the Server continue to receive and process incoming information, but suppresses any outgoing information (Web pages).

The **GetReadyForPage** method implements the behavior set by the **SetNewPageDisable** method.

Example:

```

CASE EVENT()
OF EVENT:StupidQuestion
  WebServer.SetNewPageDisable(True)      !suppress outgoing information
OF EVENT:ApoLogY
  WebServer.SetNewPageDisable(False)     !enable outgoing information
END

```

See Also: [GetReadyForPage](#)

## SetNextAction (return Web page field information)

### SetNextAction, SubmitItemClass

The **SetNextItem** method sets the `CurSubmit` property to the next incoming Web page field, then returns either a NULL reference, indicating there are no more incoming items to process, or returns a valid reference to the `CurSubmit` property (a `SubmitItemClass` object), indicating the referenced object is ready to process the next incoming item.

Implementation: The `SetNextItem` method calls the `SubmitItemClass.Reset` method to set the `CurSubmit` property to the next incoming item.

When the end user enters data through the Web browser, the browser submits the data in a “request string” which the Application Broker forwards to the Server (your Web-enabled application). The request string optionally contains a list of field assignments of the form ‘field=value&field2=value2’. The `CurSubmit` property represents a single field assignment within the request string, including the field equate of the control, the new value entered in the browser, and any event associated with the control (accepted, scroll, etc.).

Return Data Type: `SubmitItemClass` reference

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE

NextSubmit      &SubmitItemClass
CurFeq         SIGNED,AUTO
Index           SIGNED,AUTO
CODE
LOOP
  NextSubmit &= SELF.Server.SetNextAction()           !return Web page field info
  IF (NextSubmit &= NULL)                             !if NULL,
    BREAK                                             ! no more fields to process
  END
  CurFeq = NextSubmit.Feq                             !otherwise, process the field
  CASE (CurFeq)
  OF FEQ:UNKNOWN
    SELF.TakeUnknownSubmit(NextSubmit)
  ELSE
    IF (SELF.GetControl(CurFeq))
      IF (NextSubmit.Event)
        IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
          SELF.Controls.ThisControl.ResetControl(NextSubmit)
        END
      END
    END
  END
END
END
END
```

See Also: `CurSubmit`, `SubmitItemClass.Reset`

## TakeEvent (process WebServerClass object events)

### TakeEvent, SIGNED

The **TakeEvent** method process all events for the WebServerClass object and returns a value indicating whether the event originated from the Client or from the Server (internally). A return value of NET:Request indicates the event originated from the Client; a return value of NET:Unknown indicates the event originated elsewhere.

Implementation:

EQUATEs for the return value are declared in ICSEVER.INC as follows:

```
NET:Unknown      EQUATE(0)
NET:Terminate    EQUATE(1)
NET:Request      EQUATE(2)
```

The TakeEvent method handles any events that originated from the Client, that is, any events forwarded to the Server by the Application Broker through the communication channel established by the Connect method.

The TakeEvent method also handles other events that may be significant to the WebServerClass object, such as EVENT:OpenWindow events (the WebServerClass object initiates a timer to support automatic shut down after an idle period), or EVENT:Timer events (the WebServerClass object monitors timer events to support automatic shut down after an idle period). This does not interfere with any other timer events and processing for the Window.

Return Data Type:

**SIGNED**

Example:

```
WebWindowClass.TakeEvent      PROCEDURE

CODE
IF (SELF.Server.Active)
CASE SELF.Server.TakeEvent()
OF NET:Request
  SELF.TakeRequest
END
CASE (EVENT())
OF EVENT:NewPage
  IF (SELF.Server.GetReadyForPage())
    SELF.TakeCreatePage
  END
OF EVENT:OpenWindow
  SELF.ResetFromControls
END
WebActiveFrame.TakeEvent
IF (SELF.Server.GetReadyForPage())
  POST(EVENT:NewPage)
END
END
```

## TakePageSent (prepare WebServerClass object for next page)

---

### TakePageSent

The **TakePageSent** method prepares the `WebServerClass` object to process the next request from the Client (browser).

Implementation: The `TakePageSent` method reinitializes some internal flags.

Example:

```
WebWindowClass.TakeCreatePage    PROCEDURE

Client          &WebClientManagerClass
Filename        CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client

SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage())
    Filename = SELF.Files.GetFilename(Content:Html)
    IF (SELF.Server.GetRequestedWholePage())
        Client.NextHtmlPage
        SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
    ELSE
        SELF.CreateDummyHtmlPage(SELF.HtmlTarget, Filename)
    END
    Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
ELSE
    Client.Js1.OpenChannel(SELF.GetTargetSecurity(), SELF.Files)
    SELF.CreateJs1Data(Client.Js1)
    Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()                !reset for next page
```

## ShutdownClass Methods

The WebServerClass uses the ShutdownClass to shut down other IBC Library objects when the Server (Web-enabled application) undergoes a normal shut down, including end user request and time outs.

The ShutdownClass is an abstract class. Its methods are not implemented; rather they are placeholders for derived classes. The ShutdownClass contains the methods listed below.

### Close (a virtual shut down the Web-enabled application)

#### Close, VIRTUAL

The **Close** method is a virtual placeholder to shut down the Web-enabled application, including other IBC Library objects.

Example:

```

MyWebPgm      PROGRAM
!data declarations
Broker        BrokerClass           !declare Broker object
HtmlManager   HtmlClass             !declare HtmlManager object
JavaEvents    Js1EventsClass        !declare Js1Events object
WebServer     WebServerClass        !declare WebServer object
WebFilesManager WebFilesClass      !declare WebFilesManager object
ShutdownManager CLASS(ShutdownClass) !declare ShutdownManager object
Close         PROCEDURE,VIRTUAL
                END
ICServerWin   WINDOW,AT(-100,-100,0,0) !declare an invisible window
                END

CODE
WebFilesManager.Init(1, '')           !initialize WebFilesManager object
JavaEvents.Init                       !initialize JavaEvents object
Broker.Init('TREE', WebFilesManager) !initialize Broker object
HtmlManager.Init(WebFilesManager)     !initialize HtmlManager object
WebServer.Init(Broker,ShutdownManager,,600,,WebFilesManager)!init Shutdown object
OPEN(ICServerWin)                    !open invisible window
ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect                 !set up communication w/ App Broker
    Main                              !run as usual
    BREAK
  END
END
ShutdownManager.Close                 !shut down IBC Library objects

ShutdownManager.Close
CODE
WebServer.Kill                       !shut down WebServer object
HtmlManager.Kill                     !shut down HtmlManager object
Broker.Kill()                         !shut down Broker object
JavaEvents.Kill                      !shut down JavaEvents object
WebFilesManager.Kill                 !shut down WebFilesManager object

```

# ***WEB CLIENT MANAGER CLASS***

<b>Overview</b>	<b>93</b>
WebClientManagerClass Concepts .....	93
Relationship to Other Internet Builder Classes .....	93
Internet Connect Template Implementation .....	93
Source Files .....	93
<b>WebClientManagerClass Properties</b>	<b>94</b>
Broker (BrokerClass object) .....	94
Browser (BrowserManagerClass object) .....	94
IP (client IP address) .....	95
Jsl (JslManagerClass object) .....	95
<b>WebClientManagerClass Methods</b>	<b>96</b>
Functional Organization—Expected Use .....	96
Feq2Id (return HTML control Id) .....	97
Init (initialize the WebClientManagerClass object) .....	98
Kill (shut down the WebClientManagerClass object) .....	99
NextHtmlPage (prepare for next HTML page) .....	100
TakeFile (send HTML code or JSL data) .....	100
TakeHtmlPage (send HTML code) .....	101
TakeJslData (send JSL data) .....	103
TakeUnauthorized (send access denied page) .....	105



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### WebClientManagerClass Concepts

---

The WebClientManagerClass helps the WebWindowClass generate appropriate HTML code and JSL data by providing accurate information about the Client's browser. The WebClientManagerClass contains current information about the Client Web browser, Java Support Library, and IP address.

The WebClientManagerClass also communicates with the BrokerClass on behalf of the WebWindowClass. The BrokerClass instantiates and manages a single WebClientManagerClass object.

### Relationship to Other Internet Builder Classes

---

The WebClientManagerClass is derived from the WebDataSinkClass. The WebDataSinkClass is an abstract class whose methods are not defined. The WebDataSinkClass does provides a reference to all its derived classes.

The BrokerClass creates and manages the WebClientManagerClass object to represent the Client computer.

### Internet Connect Template Implementation

---

The BrokerClass creates and manages a single global WebClientManagerClass object. The template generated code does not directly reference the WebClientManagerClass object.

### Source Files

---

The WebClientManagerClass source code is installed by default to the \LIBSRC folder. The specific WebClientManagerClass files and their respective components are:

ICCLIENT.INC

WebClientManagerClass declarations

## *WebClientManagerClass Properties*

### **Broker (BrokerClass object)**

---

#### **Broker &BrokerClass, PROTECTED**

The **Broker** property is a reference to the BrokerClass object that represents the Application Broker. The Broker property handles the transmission of HTML code and Java Support Library (JSL) data from the Server (Web-enabled application) to the Application Broker.

This property is PROTECTED, therefore, it can only be referenced by a WebClientManagerClass method, or a method in a class derived from WebClientManagerClass.

Implementation: The Init method sets the value of the Broker property.

See Also: Init

### **Browser (BrowserManagerClass object)**

---

#### **Browser &BrowserManagerClass**

The **Browser** property is a reference to the BrowserManagerClass object that represents the Client's Web browser within the Server (Web-enabled application). The Browser property contains information about the specific features the Web browser supports.

The HtmlClass object, HttpClass object, and WebWindowClass object all modify their output based on the contents of the Browser property and the specific features the Web browser supports.

Implementation: The Init method sets the value of the Browser property.

See Also: Init

## IP (client IP address)

---

IP	CSTRING(255)
----	--------------

The **IP** property contains the Client's IP address. The `WebClientManagerClass` does not use this property, but makes it available for other code that may need it.

Implementation: The `BrokerClass.Init` method sets the value of the IP property. The value comes from the Application Broker that launches the Server (Web-enabled application).

See Also: `BrokerClass.Init`

## Jsl (JslManagerClass object)

---

Jsl	&JslManagerClass
-----	------------------

The **Jsl** property is a reference to the `JslManagerClass` that represents the Java Support Library (JSL). The Jsl property allows dynamic updates to the contents of an HTML page without refreshing the entire page.

Implementation: The `Init` method instantiates a `JslManagerClass` object for the Jsl property.

See Also: `Init`

## ***WebClientManagerClass Methods***

The WebClientManagerClass contains the methods listed below.

### **Functional Organization—Expected Use**

---

As an aid to understanding the WebClientManagerClass, it is useful to organize its methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### **Primary Interface Methods**

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize the WebClientManagerClass object
Kill	shut down the WebClientManagerClass object

##### **Mainstream Use:**

-none-

##### **Occasional Use:**

Feq2Id	return HTML control Id
NextHtmlPage	prepare for next HTML page

#### **Virtual Methods**

Typically you will not call these methods directly—the Primary Interface methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

TakeFile	send HTML code or JSL data
TakeHtmlPage	send HTML code
TakeJslData	send JSL data
TakeUnauthorized	send access denied page

## Feq2Id (return HTML control Id)

### Feq2Id( *control* ), UNSIGNED, VIRTUAL

#### **Feq2Id**

Returns the HTML control Id for the corresponding Clarion WINDOW or REPORT control.

#### *control*

An integer variable, constant, EQUATE, or expression containing a Clarion control number. This is typically the Field Equate label of the control.

The **Feq2Id** method returns the HTML control Id for the corresponding Clarion WINDOW or REPORT control.

Feq2Id is a VIRTUAL method so that other base class methods can directly call the Feq2Id virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

The HtmlClass.WriteAppletHeader calls the Feq2Id method to generate unique control Ids within the generated HTML.

#### Example:

```
MyHtmlClass.WriteAppletHeader PROCEDURE |
    (SIGNED Feq,STRING ClassName,SIGNED Width,SIGNED Height)

Id    SIGNED,AUTO

CODE
Id = SELF.Client.Feq2Id(Feq)
SELF.WriteAppletHeader(Feq, '_X' & Id, ClassName, Width, Height)
```

#### See Also:

**HtmlClass.WriteAppletHeader**

## Init (initialize the WebClientManagerClass object)

---

**Init**( *broker*, *browser* )

<b>Init</b>	Initializes the WebClientManagerClass object.
<i>broker</i>	The label of the BrokerClass object that handles the transmission of HTML code and Java Support Library (JSL) data from the Server (Web-enabled application) to the Application Broker.
<i>browser</i>	The label of the BrowserManagerClass object that contains information about the specific features the Client's Web browser supports.

The **Init** method initializes the WebClientManagerClass object.

Implementation:       The Init method sets the value of the Broker, Browser, and Jsl properties.

Example:

```
BrokerClass.SetClient           PROCEDURE
CODE
IF (SELF.CurClient &= NULL)
  SELF.CurClient &= NEW WebClientManagerClass       !instantiate client object
  SELF.CurClient.Init(SELF, UnknownBrowser)       !initialize client object
END
SELF.SetClientBrowser
```

See Also:                **Broker, Browser, Jsl**

## Kill (shut down the WebClientManagerClass object)

---

### Kill

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Example:

```
BrokerClass.Kill          PROCEDURE
CODE
SELF.CloseChannel
IF (NOT (SELF.Http &= NULL))
    SELF.Http.Kill
    DISPOSE(SELF.Http)
END
IF (NOT SELF.CurClient &= NULL)
    SELF.CurClient.Kill          !shut down client object
    DISPOSE(SELF.CurClient)
END
```

## NextHtmlPage (prepare for next HTML page)

---

### NextHtmlPage

The **NextHtmlPage** method prepares the `WebClientManagerClass` object to process a new HTML page.

Implementation: The `NextHtmlPage` method increments a counter used to generate unique IDs for HTML controls.

Example:

```
MyWebWindowClass.TakeCreatePage PROCEDURE

Client      &WebClientManagerClass
Filename    CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client

SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF SELF.Server.GetSendWholePage()           !if sending whole page
  Filename = SELF.Files.GetFilename(Content:Html)
  Client.NextHtmlPage                       !prepare Client object for new page
  SELF.CreateHtmlPage(SELF.HtmlTarget,Filename) !WebWindow object creates new page
  Client.TakeHtmlPage(Filename,SELF.GetTargetSecurity(),FALSE) !Client takes it
ELSE
  Client.Jsl.OpenChannel(SELF.GetTargetSecurity(),SELF.Files)
  SELF.CreateJslData(Client.Jsl)
  Client.Jsl.CloseChannel
END
SELF.Server.TakePageSent()                 !prepare Server object for next page
```

## TakeFile (send HTML code or JSL data)

---

### TakeFile( *filename*, *secure*, *dontmove* ), VIRTUAL

<b>TakeFile</b>	Requests the <code>BrokerClass</code> object to send HTML code or JSL data to the Client browser.
<i>filename</i>	A string constant, variable, <code>EQUATE</code> , or expression naming the file containing the HTML code or JSL data to transmit.
<i>secure</i>	An integer constant, variable, <code>EQUATE</code> , or expression indicating whether to transmit over a secure channel or public channel. A value of <code>Secure:Full</code> indicates a secure channel; a value of <code>Secure:None</code> indicates a public channel; and a value of <code>Secure:Default</code> indicates the default channel.
<i>dontmove</i>	An integer constant, variable, <code>EQUATE</code> , or expression indicating ...

The **TakeFile** method requests the BrokerClass object to send HTML code or JSL data to the Client browser.

TakeFile is a VIRTUAL method so that other base class methods can directly call the TakeFile virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The TakeFile method calls the BrokerClass.TakeFile method.

EQUATEs for the secure parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

See Also: BrokerClass.TakeFile

## TakeHtmlPage (send HTML code)

**TakeHtmlPage**( *filename*, *secure*, *dontmove* ), VIRTUAL

<b>TakeHtmlPage</b>	Requests the BrokerClass object to send an HTML page to the Client browser.
<i>filename</i>	A string constant, variable, EQUATE, or expression naming the file containing the HTML code to transmit.
<i>secure</i>	An integer constant, variable, EQUATE, or expression indicating whether to transmit over a secure channel or public channel. A value of Secure:Full indicates a secure channel; a value of Secure:None indicates a public channel; and a value of Secure:Default indicates the default channel.
<i>dontmove</i>	An integer constant, variable, EQUATE, or expression indicating ...

The **TakeHtmlPage** method requests the BrokerClass object to send an HTML page to the Client browser.

TakeHtmlPage is a VIRTUAL method so that other base class methods can directly call the TakeHtmlPage virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: TakeHtmlPage calls the BrokerClass.TakeHtmlPage method.

EQUATEs for the secure parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE

```

```
None      EQUATE
Full      EQUATE
Last      EQUATE(Secure:Full)
          END
```

Example:

```

MyWebWindowClass.TakeCreatePage PROCEDURE

Client      &WebClientManagerClass
Filename    CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client

SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF SELF.Server.GetSendWholePage()          !if sending whole page
  Filename = SELF.Files.GetFilename(Content:Html)
  Client.NextHtmlPage                      !prepare Client object for new page
  SELF.CreateHtmlPage(SELF.HtmlTarget,Filename) !WebWindow object creates new page
  Client.TakeHtmlPage(Filename,SELF.GetTargetSecurity(),FALSE) !Client takes it
ELSE
  Client.Jsl.OpenChannel(SELF.GetTargetSecurity(),SELF.Files)
  SELF.CreateJslData(Client.Jsl)
  Client.Jsl.CloseChannel
END
SELF.Server.TakePageSent()                  !prepare Server object for next page

```

See Also: `BrokerClass.TakeHtmlPage`

## TakeJslData (send JSL data)

**TakeJslData**( *filename*, *secure* ), VIRTUAL

<b>TakeJslData</b>	Requests the BrokerClass object to send JSL data to the Client browser.
<i>filename</i>	A string constant, variable, EQUATE, or expression naming the file containing the JSL data to send.
<i>secure</i>	An integer constant, variable, EQUATE, or expression indicating whether to transmit over a secure channel or public channel. A value of Secure:Full indicates a secure channel; a value of Secure:None indicates a public channel; and a value of Secure:Default indicates the default channel.

The **TakeJslData** method requests the BrokerClass object to send JSL data to the Client browser.

TakeJslData is a VIRTUAL method so that other base class methods can directly call the TakeJslData virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: TakeJslData calls the BrokerClass.TakeJslData method.

EQUATES for the secure parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE

```

```
None      EQUATE
Full      EQUATE
Last      EQUATE(Secure:Full)
          END
```

See Also: **BrokerClass.TakeJsldata**

## TakeUnauthorized (send access denied page)

### TakeUnauthorized( *filename*, *secure* ), VIRTUAL

**TakeUnauthorized** Requests the BrokerClass object to send an “access denied” page to the Client browser.

*filename* A string constant, variable, EQUATE, or expression naming the file containing the “access denied” page to transmit.

*secure* An integer constant, variable, EQUATE, or expression indicating whether to transmit over a secure channel or public channel. A value of Secure:Full indicates a secure channel; a value of Secure:None indicates a public channel; and a value of Secure:Default indicates the default channel.

The **TakeUnauthorized** method requests the BrokerClass object to send an “access denied” page to the Client browser.

TakeUnauthorized is a VIRTUAL method so that other base class methods can directly call the TakeUnauthorized virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

TakeUnauthorized calls the BrokerClass.TakeUnauthorized method.

EQUATEs for the secure parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

Example:

```

MyWebWindowClass.TakeCreatePage      PROCEDURE

Client          &WebClientManagerClass
Filename        CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client

SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
  IF (SELF.Authorize AND NOT SELF.GetAuthorized())
    Filename = SELF.Files.GetFilename(Content:Unauthorized)
    SELF.CreateUnauthorizedPage(SELF.HtmlTarget, Filename)
    Client.TakeUnauthorized(Filename,SELF.GetTargetSecurity()) !access denied
    SELF.AuthorizeFailed = TRUE
    Post(EVENT:CloseWindow)
  ELSE
    Filename = SELF.Files.GetFilename(Content:Html)
    IF (SELF.Server.GetRequestedWholePage())
      Client.NextHtmlPage
      SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
    ELSE
      SELF.CreateDummyHtmlPage(SELF.HtmlTarget, Filename)
    END
    Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
  END
ELSE
  Client.Js1.OpenChannel(SELF.GetTargetSecurity(), SELF.Files)
  SELF.CreateJs1Data(Client.Js1)
  Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()

```

See Also: **BrokerClass.TakeUnauthorized**

# ***BROWSER MANAGER CLASS***

<b>Overview</b>	<b>109</b>
BrowserManagerClass Concepts .....	109
Relationship to Other Internet Builder Classes .....	109
Internet Connect Template Implementation .....	109
Source Files .....	109
<b>BrowserManagerClass Properties</b>	<b>110</b>
Kind (browser type) .....	110
SetNoCache (external cache control support) .....	110
SupportsStyleSheets (style sheet support) .....	111
SubmitFromJava (applet communication mode) .....	111
<b>BrowserManagerClass Methods</b>	<b>112</b>
Init (initialize the BrowserManagerClass object) .....	112



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### BrowserManagerClass Concepts

---

The BrowserManagerClass object contains information about the specific features the Client's Web browser supports. The information is available to any other code that needs it.

### Relationship to Other Internet Builder Classes

---

The BrokerClass creates several instances of the BrowserManagerClass—one for each major type of browser. When the Client browser identifies itself (with HTTP header information), the BrokerClass references the particular BrowserManagerClass object that represents the Client's browser.

The HtmlClass object uses the BrowserManagerClass object to custom tailor its generated HTML code to the specific browser the Client is running.

### Internet Connect Template Implementation

---

The BrokerClass creates and manages instances of the BrowserManagerClass as needed. Therefore, the Internet Connect Template generated code does not directly reference the BrowserManagerClass object.

### Source Files

---

The BrowserManagerClass source code is installed by default to the \LIBSRC folder. The BrowserManagerClass declarations reside in the following files and their method definitions reside in the corresponding .CLW files.

ICCLIENT.INC

BrowserManagerClass

## BrowserManagerClass Properties

### Kind (browser type)

---

**Kind**    **BYTE**

The **Kind** property indicates the major category the represented Web browser falls into. Valid Web browser categories include:

- Internet Explorer version 3.0
- Internet Explorer version 4.0
- NetScape version 3
- Mozilla version 4
- Unknown

The HtmlClass object uses the Kind property to configure generated HTML to the Client's particular browser.

Implementation:    The Init method sets the value of the Kind property. The HtmlClass.WriteRefreshTimer method alters its output based on the value of the Kind property.

EQUATEs for the Kind property are declared in ICCLIENT.INC as follows:

```

                                ITEMIZE, PRE(BROWSER)
IE30                            EQUATE
IE40                            EQUATE
NetScape3                       EQUATE
Mozilla4                        EQUATE
Unknown                         EQUATE
                                END

```

See Also:            Init

### SetNoCache (external cache control support)

---

**SetNoCache**    **BYTE**

The **SetNoCache** property indicates whether the represented browser accepts external requests to suppress caching. A value of one (1) indicates the browser accepts external (HTTP) requests to suppress caching; a value of zero (0) indicates the browser caching mode cannot be set externally.

The HttpClass object uses the SetNoCache property to configure generated HTTP to the Client's particular browser.

Implementation:    The Broker.Init method sets the value of the SetNoCache property. The HttpClass object alters its output based on the value of the SetNoCache property.

See Also:            Broker.Init

## SupportsStyleSheets (style sheet support)

---

### SupportsStyleSheets    BYTE

The **SupportsStyleSheets** property indicates whether the represented browser supports style sheets. A value of one (1) indicates the browser supports style sheets; a value of zero (0) indicates the browser does not support style sheets.

Implementation:    The `Init` method sets the value of the `SupportsStyleSheets` property. The `HtmlClass` and the `WebWindowClass` alter their output based on the value of the `SupportsStyleSheets` property.

See Also:    `Init`

## SubmitFromJava (applet communication mode)

---

### SubmitFromJava    BYTE

The **SubmitFromJava** property indicates how the represented browser channels Java applet requests to the Web server. A value of one (1) indicates the browser's applets communicate directly with the Web server; a value of zero (0) indicates the browser's applets communicate indirectly with the Web server, by going through the browser's usual channels.

The `HtmlClass` object uses the `SubmitFromJava` property to configure generated HTML to the represented browser.

Implementation:    The `Init` method sets the value of the `SubmitFromJava` property. The `HtmlClass.WriteJavaScript` method alters its output based on the value of the `SubmitFromJava` property.

See Also:    `Init`

## BrowserManagerClass Methods

The BrowserManagerClass contains only one method.

### Init (initialize the BrowserManagerClass object)

**Init**( *kind, style sheets, applet submit* ), VIRTUAL

<b>Init</b>	Initializes the BrowserManagerClass object.
<i>kind</i>	An integer constant, variable, EQUATE, or expression indicating the major category the Web browser falls into. Valid Web browser categories include Internet Explorer 3 and 4, NetScape 3, Mozilla 4, and other.
<i>style sheets</i>	An integer constant, variable, EQUATE, or expression indicating whether the browser supports style sheets. A value of one (1) indicates the browser supports style sheets; a value of zero (0) indicates the browser does not support style sheets.
<i>applet submit</i>	An integer constant, variable, EQUATE, or expression indicating how the browser channels Java applet requests to the Web server. A value of one (1) indicates the browser's applets communicate directly with the Web server; a value of zero (0) indicates the browser's applets communicate indirectly with the Web server, by going through the browser's usual channels.

The **Init** method initializes the BrowserManagerClass object.

Implementation:

The Init method sets the values of the Kind, SupportsStyleSheets, and SubmitFromJava properties.

EQUATEs for the *kind* parameter are declared in ICCLIENT.INC as follows:

```

ITEMIZE,PRE(BROWSER)
IE30      EQUATE
IE40      EQUATE
NetScape3 EQUATE
Mozilla4  EQUATE
Unknown   EQUATE
END

```

Example:

```
MyBrokerClass.Init PROCEDURE(STRING ProgramName, WebFilesClass Files)
```

```
CODE
```

```
IE30.Init(BROWSER:IE30,TRUE,FALSE)           !init Internet Explorer 3.0 browser  
IE40.Init(BROWSER:IE40,TRUE,FALSE)           !init Internet Explorer 4.0 browser  
NetScape3x.Init(BROWSER:NetScape3,FALSE,TRUE) !init NetScape 3x browser  
NetScape3x.SetNoCache = TRUE  
Mozilla4.Init(BROWSER:Mozilla4,FALSE,TRUE)   !init Mozilla 4 browser  
Mozilla4.SetNoCache = TRUE  
UnknownBrowser.Init(BROWSER:Unknown,FALSE,FALSE)!init generic Java enabled browser  
  
SELF.SetClient                               !point Client to appropriate browser
```

See Also:

**Kind, SupportsStyleSheets, SubmitFromJava**



# WEB FRAME CLASS

<b>Overview</b>	<b>117</b>
WebFrameClass Concepts .....	117
Relationship to Other Internet Builder Classes .....	117
Internet Connect Template Implementation .....	117
Source Files .....	117
Conceptual Example .....	118
<b>WebFrameClass Properties</b>	<b>119</b>
FrameWindow (MDI frame window) .....	119
MenubarFeq (menubar control number) .....	119
ToolbarFeq (Toolbar control number) .....	119
<b>WebFrameClass Methods</b>	<b>120</b>
CopyControlsToWindow (a virtual to copy frame controls to child window) ...	120
CopyControlToWindow (copy frame control to child window) .....	122
GetMenubarFeq (return menubar control number) .....	123
GetToolbarFeq (return Toolbar control number) .....	124
TakeEvent (a virtual to handle menu and toolbar events) .....	125



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

## WebFrameClass Concepts

---

Typically, your Web-enabled application instantiates a `WebFrameClass` object to represent the mergeable controls on the APPLICATION frame.

The `WebFrame Class` provides methods to “merge” these global toolbar and menu controls onto the Web pages representing the application’s child windows. Finally, `WebFrame Class` provides methods to handle any events associated with the merged controls.

## Relationship to Other Internet Builder Classes

---

The `WebFrameClass` is independent of other Internet Builder Classes.

## Internet Connect Template Implementation

---

The Internet Connect Templates generate code to derive a `WebFrameClass` object to represent the mergeable controls on the APPLICATION frame.

The derived `WebFrameClass` object is called `MainFrame`. The Internet Connect Templates generate virtual methods within the `MainFrame` class to simulate standard Windows menu and toolbar merging behavior on the Web pages generated by the Server (your Web enabled application).

## Source Files

---

The `WebWindowClass` source code is installed by default to the `\LIBSRC` folder. The source files and their components are as follows:

`ICWINDOW.INC` `WebFrameClass` class declarations

`ICWINDOW.CLW` `WebFrameClass` method definitions

## Conceptual Example

---

The following example uses the `WebFrameClass` object to simulate merging menus onto a “child” Web page.

```

PROGRAM
!data
MainFrame          CLASS(WebFrameClass)
CopyControlsToWindow  PROCEDURE(*WebWindowClass,BYTE,BYTE),VIRTUAL
FrameWindow        &WINDOW
                    END

CODE
!program code

MainFrame.CopyControlsToWindow PROCEDURE|
    (*WebWindowClass OwnerWindow, BYTE MergeMenu, BYTE MergeTool)
CODE
IF MergeMenu
    SELF.CopyControlToWindow(OwnerWindow, ?Exit)
    SELF.CopyControlToWindow(OwnerWindow, ?BrowseCustomer)
    SELF.CopyControlToWindow(OwnerWindow, ?BrowseProduct)
    SELF.CopyControlToWindow(OwnerWindow, ?BrowseState)
    SELF.CopyControlToWindow(OwnerWindow, ?Print)
    SELF.CopyControlToWindow(OwnerWindow, ?PrintInvoice)
    SELF.CopyControlToWindow(OwnerWindow, ?PrintMailingLabels)
    SELF.CopyControlToWindow(OwnerWindow, ?PrintPriceList)
    SELF.CopyControlToWindow(OwnerWindow, ?ReadMe)
END

BrowseCustomer  PROCEDURE
!data
CODE
!procedure code
PrepareProcedure ROUTINE
IF (WebServer.Active)
    IC:CurFrame &= GetWebActiveFrame()
    IC:CurFrame.CopyControlsToWindow(WebWindow, TRUE, TRUE)
END

```

## WebFrameClass Properties

The WebFrameClass contains the following properties.

### FrameWindow (MDI frame window)

---

#### FrameWindow &WINDOW

The **FrameWindow** property is a reference to the program's APPLICATION (main window). The WebFrameClass uses the FrameWindow property to reference the program's main window.

### MenubarFeq (menubar control number)

---

#### MenubarFeq SIGNED

The **MenubarFeq** property contains the control number (field equate) of the MENUBAR. The WebFrameClass uses this property to refer to the WINDOW's MENUBAR structure.

Implementation: The GetMenubarFeq method returns the value of the MenubarFeq property.

See Also: GetMenubarFeq

### ToolbarFeq (Toolbar control number)

---

#### ToolbarFeq SIGNED

The **ToolbarFeq** property contains the control number (field equate) of the TOOLBAR. The WebFrameClass uses this property to refer to the WINDOW's TOOLBAR structure.

Implementation: The GetToolbarFeq method returns the value of the ToolbarFeq property.

See Also: GetToolbarFeq

## WebFrameClass Methods

The WebFrameClass contains the methods listed below.

### CopyControlsToWindow (a virtual to copy frame controls to child window)

**CopyControlsToWindow**( *window object, merge menu, merge toolbar*), **VIRTUAL**

#### CopyControlsToWindow

Is a virtual placeholder method to copy controls from the application frame to another window.

*window object* The label of the WebWindowClass object that represents the WINDOW that receives the copied controls.

*merge menu* An integer constant, variable, EQUATE, or expression indicating whether to copy menu controls. A value of one (1) copies menu controls; a value of zero (0) does not copy menu controls.

*merge toolbar* An integer constant, variable, EQUATE, or expression indicating whether to copy toolbar controls. A value of one (1) copies toolbar controls; a value of zero (0) does not copy toolbar controls.

The **CopyControlsToWindow** method is a virtual placeholder method to copy controls from the application frame to another (child) window. This allows the simulation of standard Windows (menu and toolbar) merging behavior within Web pages as seen through the Client browser.

CopyControlsToWindow is a VIRTUAL method so that other base class methods can directly call the CopyControlsToWindow virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

The CopyControlsToWindow method does nothing. It is a virtual placeholder for derived classes. In particular, the Internet Connect Templates generate a custom CopyControlsToWindow method for each Web-enabled application. The Internet Connect Template generated CopyControlsToWindow calls the CopyControlToWindow method for each merged control.

Example:

```
BrowseCustomer PROCEDURE
!data
CODE
!procedure code
PrepareProcedure ROUTINE
IF (WebServer.Active)
    IC:CurFrame &= GetWebActiveFrame()
    IC:CurFrame.CopyControlsToWindow(WebWindow, TRUE, TRUE)
END
```

See Also: [CopyControlToWindow](#)

## CopyControlToWindow (copy frame control to child window)

**CopyControlToWindow( *window object*, *control* ), PROTECTED**

### CopyControlToWindow

Copies the specified *control* from the application frame to another window.

*window object* The label of the WebWindowClass object that represents the WINDOW that receives the copied *control*.

*control* An integer constant, variable, EQUATE, or expression containing the number (field equate) of the control to copy.

The **CopyControlToWindow** method CREATES a copy of the specified frame *control* on the current (child) WINDOW. This allows the simulation of standard Windows (menu and toolbar) merging behavior within Web pages as seen through the Client browser.

This method is PROTECTED, therefore, it can only be called by a WebFrameClass method, or a method in a class derived from WebFrameClass.

Implementation:

The CopyControlToWindow method CREATES a copy of the specified frame *control* on the current (child) WINDOW. The CopyControlToWindow method calls the WebWindowClass.AddControl method for the CREATED control.

Example:

```
MainFrame.CopyControlsToWindow PROCEDURE|
    (*WebWindowClass OwnerWindow, BYTE MergeMenu, BYTE MergeTool)
CODE
IF MergeMenu
    SELF.CopyControlToWindow(OwnerWindow, ?Exit)
    SELF.CopyControlToWindow(OwnerWindow, ?BrowseCustomer)
    SELF.CopyControlToWindow(OwnerWindow, ?BrowseProduct)
    SELF.CopyControlToWindow(OwnerWindow, ?BrowseState)
    SELF.CopyControlToWindow(OwnerWindow, ?Print)
    SELF.CopyControlToWindow(OwnerWindow, ?PrintInvoice)
    SELF.CopyControlToWindow(OwnerWindow, ?PrintMailingLabels)
    SELF.CopyControlToWindow(OwnerWindow, ?PrintPriceList)
    SELF.CopyControlToWindow(OwnerWindow, ?ReadMe)
END
```

See Also:

WebWindowClass.AddControl

## GetMenubarFeq (return menubar control number)

---

### GetMenubarFeq, SIGNED, VIRTUAL

The **GetMenubarFeq** method returns the control number (field equate) of the window's MENUBAR.

GetMenubarFeq is a VIRTUAL method so that other base class methods can directly call the GetMenubarFeq virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The GetMenubarFeq method returns the value of the MenubarFeq property.

Return Data Type:     **SIGNED**

Example:

```
GetParentFeq     PROCEDURE(SIGNED Feq, *WebWindowBaseClass OwnerWindow)
ParentFeq        SIGNED,AUTO
CODE
ParentFeq = Feq{PROP:Parent}
IF (ParentFeq = 0)
CASE (Feq{PROP:Type})
OF CREATE:MENU
OROF CREATE:ITEM
  ParentFeq = OwnerWindow.GetMenubarFeq()
ELSE
  IF (Feq{PROP:intoolbar})
    ParentFeq = OwnerWindow.GetToolbarFeq()
  ELSE
    ParentFeq = FEQ:ClientArea
  END
END
END
RETURN ParentFeq
```

See Also:             **MenubarFeq**

## GetToolBarFeq (return Toolbar control number)

---

### GetToolBarFeq, SIGNED, VIRTUAL

The **GetToolBarFeq** method returns the control number (field equate) of the window's TOOLBAR.

GetToolBarFeq is a VIRTUAL method so that other base class methods can directly call the GetToolBarFeq virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The GetToolBarFeq method returns the value of the ToolbarFeq property.

Return Data Type:     **SIGNED**

Example:

```
GetParentFeq     PROCEDURE(SIGNED Feq, *WebWindowBaseClass OwnerWindow)
ParentFeq        SIGNED,AUTO
CODE
ParentFeq = Feq{PROP:Parent}
IF (ParentFeq = 0)
CASE (Feq{PROP:Type})
OF CREATE:MENU
OROF CREATE:ITEM
  ParentFeq = OwnerWindow.GetToolBarFeq()
ELSE
  IF (Feq{PROP:intoolbar})
    ParentFeq = OwnerWindow.GetToolBarFeq()
  ELSE
    ParentFeq = FEQ:ClientArea
  END
END
END
RETURN ParentFeq
```

See Also:             **ToolBarFeq**

## TakeEvent (a virtual to handle menu and toolbar events)

### TakeEvent, SIGNED, VIRTUAL, PROC

The **TakeEvent** method is a virtual placeholder method to handle any global menu and toolbar events from a child window. This allows the simulation of standard Windows (menu and toolbar) merging behavior within Web pages as seen through the Client browser.

The TakeEvent return value indicates whether the calling entity should CYCLE to the top, BREAK out of, or continue through the current ACCEPT loop.

TakeEvent is a VIRTUAL method so that other base class methods can directly call the TakeEvent virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method has the PROC attribute so you can call it like a PROCEDURE and ignore the return value.

Implementation:

The WebWindowClass.TakeEvent method calls the WebFrameClass.TakeEvent method. The Internet Connect Templates derive a WebFrameClass.TakeEvent method for each Web-enabled application.

Return value EQUATEs are declared in ICWINDOW.INC as follows:

```
REPLY:NONE           EQUATE(0)
REPLY:CYCLE          EQUATE(1)
REPLY:BREAK          EQUATE(2)
```

Return Data Type:

**SIGNED**

Example:

```
MainFrame.TakeEvent FUNCTION
FirstIteration          SIGNED(1)
CODE
LOOP
  IF (NOT FirstIteration)
    RETURN REPLY:CYCLE
  END
  FirstIteration = FALSE
  CASE FIELD()
  OF ?BrowseCustomer
    CASE EVENT()
    OF EVENT:Accepted
      START(BrowseCustomer,050000)
    END
  END
  RETURN REPLY:NONE
END
RETURN REPLY:BREAK
```

See Also:

**WebWindowClass.TakeEvent**



# WEB WINDOW CLASSES

<b>Overview</b>	<b>131</b>
WebWindowClass Concepts .....	131
Relationship to Other Internet Builder Classes .....	132
Internet Connect Template Implementation .....	132
Source Files .....	133
<b>WebControlListClass Methods</b>	<b>134</b>
Functional Organization—Expected Use .....	134
AddControlsToLayout (set controls for HTML generation) .....	135
CreateHtml (generate HTML for controls) .....	136
Init (initialize the WebControlListClass object) .....	138
Kill (shut down the WebControlListClass object) .....	138
SetParentDefaults (set a control's children) .....	139
<b>WebWindowBaseClass Properties</b>	<b>140</b>
AllowJava (generate or suppress JavaScript) .....	140
Background (window background color) .....	140
BackImage (window wallpaper) .....	140
BorderWidth (Web page border width) .....	141
CloseImage (close button graphic) .....	141
CreateCaption (include a titlebar on the Web page) .....	141
CreateClose (include a close button on the Web page) .....	142
CreateToolbar (include a toolbar on the Web page) .....	142
DefaultButton (enter key button) .....	143
DefaultButtonNeeded (simulate default button) .....	143
DisabledAction (default HTML for disabled controls) .....	144
Files (WebClientManagerClass object) .....	144
FormatBorderWidth (HTML table cell border width) .....	145
GroupBorderWidth (group box border width) .....	145
HelpDocument (HTML help document) .....	145
HelpEnabled (HTML help enabled flag) .....	146
HelpRelative (remote or local help document) .....	146
HelpStyle (HTML help style) .....	146
HtmlOption (window/control scaling information) .....	147
IsSplash (splash screen flag) .....	147
MenubarFeq (menubar control number) .....	148

MenuBarType (menu placement) .....	148
OptionBorderWidth (option box border width).....	148
PageBackground (web page background color) .....	149
PageImage (web page wallpaper) .....	149
Server (WebServerClass object) .....	149
SheetBorderWidth (sheet border width) .....	150
SnapX (horizontal control alignment factor).....	150
SnapY (vertical control alignment factor) .....	150
TimerAction (time released browser action) .....	151
TimerDelay (time interval for browser action).....	151
ToolBarFeq (toolbar control number) .....	152
<b>WebWindowBaseClass Methods</b> .....	<b>153</b>
Functional Organization—Expected Use .....	153
CreateChildHtml (create HTML for control's children) .....	153
GetBackgroundColor (return Web window background color) .....	154
GetBackgroundImage (return Web window wallpaper) .....	154
GetControl (return control information).....	155
GetCreateClose (return close button flag) .....	155
GetChildren (return all child controls) .....	156
GetFirstChild (return first child control) .....	157
GetHelpHandler (return HTML to show help document) .....	157
GetHelpReference (return HTML help document reference) .....	158
GetHelpTarget (return HTML HREF for help display) .....	158
GetMenuBarFeq (return menubar control number) .....	158
GetPageImage (return Web page wallpaper) .....	159
GetShowMenuBar (return menubar include/omit flag) .....	159
GetShowToolBar (return toolbar include/omit flag) .....	159
GetToolBarFeq (return Toolbar control number) .....	160
GetToolBarMode (return toolbar entity) .....	160
GetWebActiveFrame (return WebFrameClass reference) .....	161
<b>WebWindowClass Properties</b> .....	<b>162</b>
Authorize (require username and password) .....	162
AuthorizeArea (name of password protected Web page) .....	163
HtmlTarget (HtmlClass object) .....	163
IsCentered (center or left-justify web window) .....	163
IsSecure (public or secure channel).....	164
SentHtml (first time process).....	164

<b>WebWindowClass Methods</b>	<b>165</b>
Functional Organization—Expected Use .....	165
AddControl (add control information) .....	167
AddControlsToLayout (set controls for HTML generation) .....	169
BodyFooter (generate HTML BODY footer) .....	170
BodyHeader (generate HTML BODY header) .....	171
CreateChildHtml (create HTML for control's children) .....	172
CreateDummyHtmlPage (write empty Html page) .....	173
CreateHtmlPage (generate HTML for a window) .....	174
CreateJslData (generate Java Support Library data) .....	175
CreatePageFooter (generate HTML page footer) .....	176
CreatePageHeader (generate HTML page footer) .....	176
CreateUnauthorizedPage (create unauthorized user page) .....	177
GetAuthorized (check user authorization) .....	179
GetBackgroundColor (return Web window background color) .....	180
GetBackgroundImage (return Web page wallpaper) .....	181
GetButtonInClientArea (return button present indicator) .....	182
GetChildren (return all child controls) .....	183
GetControl (return control information) .....	184
GetControlInfo (return control reference) .....	185
GetCreateClose (return close button flag) .....	186
GetFirstChild (return first child control) .....	187
GetHelpHandler (return HTML to show help document) .....	188
GetHelpReference (return HTML help document reference) .....	189
GetHelpTarget (return HTML HREF for help display) .....	190
GetMenubarFeq (return menubar control number) .....	191
GetPageImage (return Web page wallpaper) .....	192
GetShowMenubar (return menubar include/omit flag) .....	192
GetShowToolbar (return toolbar include/omit flag) .....	193
GetTableAttributes (return window HTML <STYLE>) .....	194
GetTargetSecurity (return public or secure flag) .....	195
GetToolbarFeq (return Toolbar control number) .....	196
GetToolbarMode (return toolbar entity) .....	197
GetWebActiveFrame (return WebFrameClass reference) .....	198
Init (initialize the WebWindowClass object) .....	199
Kill (shut down the WebWindowClass object) .....	200
ResetFromControls (set control information) .....	200
SetBackground (set Web page window background) .....	201
SetCentered (center window in Web page) .....	202

SetChildDefaults (set children of each control) .....	202
SetFormatOptions (set Web page scale and alignment) .....	203
SetHelpDocument (enable single document Web page help) .....	204
SetHelpURL (enable multiple document Web page help) .....	205
SetPageBackground (set Web page background) .....	206
SetPassword (require Web page password) .....	207
SetSplash (make this a splash window) .....	208
SetTimer (set Web page timer and action) .....	209
SuppressControl (omit control from Web page) .....	210
TakeCreatePage (fill Client request for page) .....	211
TakeEvent (handle browser and ACCEPT loop events) .....	212
TakeRequest (process browser event/request) .....	213
TakeUnknownSubmit (a virtual to handle unexpected requests) .....	214
TitleContents (set browser titlebar) .....	215
ValidatePassword (verify password) .....	216

## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

## WebWindowClass Concepts

---

Typically, your Web-enabled application instantiates a local WebWindowClass object for each Window you want to display with the client browser. The WebWindowClass has two jobs:

- It “translates” a Clarion WINDOW to its HTML equivalent, control by control.
- It processes requests (submit strings) from the Client. This request processing is *in addition to* your application’s normal event processing.

### HTML Translation

The WebWindowClass contains a variety of properties and methods that determine how the translation to HTML is accomplished. For example it contains properties to specify background, menu, toolbar, and client area colors, to specify the action to take for disabled controls, any special events to generate for enabled controls, and many others.

The WebWindowClass creates and manages instances of the WebControlClass as needed to translate each control in the WINDOW. See *Web Control Classes* for more information.

### Event Processing

The WebWindowClass handles all additional events that arise when the Server is running at the request of a Client. This event handling is in addition to your application’s normal event handling. Conceptually, the WebWindowClass event processing does the following:

- 1 Handles any events generated by the Application Broker. The Application Broker generates events that tell the Server what action occurred on the Client (mouse-click, data entry, etc.), the new contents of data fields on the Client, and what information is requested by the Client.
- 2 Translates the events in item 1 to Clarion events that fall under the application’s normal event handling. For example, an EVENT:CloseWindow, or an EVENT:Accepted to a control.

- 3 After the normal event handling is done, sends the appropriate HTML code and JSL data to the Application Broker for forwarding to the Client.

## Relationship to Other Internet Builder Classes

---

The `WebWindowClass` is derived from the `WebWindowBaseClass`, which is in turn derived from the `WebControlListClass`. The `WebControlListClass` manages a list of controls and their HTML equivalents. The `WebWindowBaseClass` adds parent/child relationships between controls, plus a variety of default HTML translation properties, plus a reference to the `WebClientManagerClass` object and `WebServerClass` object.

Finally, the `WebWindowClass` adds the ability to generate HTML representing the entire window, plus security, plus the “Web” event processing for the window.

The `WebWindowClass` uses many of the other Internet Builder Classes. Therefore, if your program uses the `WebWindowClass`, it also needs these other classes. The declaration of these other classes is automatic when you `INCLUDE` the `WebWindowClass` header (`ICWINDOW.INC`) in your program’s data section.

The `WebWindowClass` also provides a variety of information about the Client, the HTML translation defaults, the parents and children of controls, etc. to the `WebControlClass` objects so they can generate appropriate HTML for the represented control.

The `WebWindowClass` uses the `WebFrameClass` to “merge” global toolbar and menu controls onto MDI windows.

## Internet Connect Template Implementation

---

The Internet Connect Templates generate code to instantiate a local `WebWindowClass` object for each window your application program displays in the client browser.

The `WebWindowClass` object is called `WebWindow`. The template generated code “sets up” the `WebWindow` object at the beginning of the procedure, “shuts down” the object at the end of the procedure, and inserts the `WebWindow` object’s event handling into the procedure’s `ACCEPT` loop.

## Source Files

---

The WebWindowClass source code is installed by default to the \LIBSRC folder. The source files and their components are as follows:

ICWINDOW.INC	WebControlListClass class declarations WebWindowBaseClass class declarations WebWindowClass class declarations
ICWINDOW.CLW	WebControlListClass method definitions WebWindowBaseClass method definitions WebWindowClass method definitions
ICWINDOW.TRN	WebWindowClass translation strings

## ***WebControlListClass Methods***

The WebControlListClass contains the methods listed below.

### **Functional Organization—Expected Use**

---

Typically, you will not use the WebControlListClass independently. Its properties and methods are designed to be inherited and used by derived classes such as the WebWindowClass. See *WebWindowClass Methods—Functional Organization* for more information.

## AddControlsToLayout (set controls for HTML generation)

**AddControlsToLayout**( *control list*, *layout object* ), **VIRTUAL**

### AddControlsToLayout

Sets the controls to include in the HTML layout and generation process.

*control list*      The label of the structure containing the list of controls to include in the HTML layout and generation process.

*layout object*      The label of the LayoutHtmlClass object that optimally arranges the controls on the generated Web page.

The **AddControlsToLayout** method sets the controls to include in the HTML layout and generation process. Any controls omitted from the generated Web page are not included in this process, for example, disabled or hidden controls may be omitted from the generated Web page.

AddControlsToLayout is a VIRTUAL method so that other base class methods can directly call the AddControlsToLayout virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

AddControlsToLayout assumes the *control list* has already been built by appropriate calls to the WebWindowClass.AddControl method, and includes only the “visible” controls from this list into the HTML layout process. The WebControlClass.GetVisible method evaluates whether a control is visible.

The *control list* parameter must name a QUEUE with a structure the same as the WebControlQueue declared in ICWINDOW.INC:

```
WebControlQueue  QUEUE,TYPE
Feq              SIGNED
ThisControl     &WebControlClass
END
```

Example:

```
WebWindowClass.AddControlsToLayout  PROCEDURE(*LayoutHtmlClass Layout, SIGNED ParentFeq)
Children                            WebControlQueue

CODE
SELF.GetChildren(Children, ParentFeq)
SELF.AddControlsToLayout(Children, Layout)
```

See Also:

WebWindowClass.AddControl, WebControlClass.GetVisible

## CreateHtml (generate HTML for controls)

**CreateHtml( *control list*, *html object*, *style*, *snapX*, *snapY* ), VIRTUAL**

<b>CreateHtml</b>	Generates HTML code to represent each visible control in the <i>control list</i> .
<i>control list</i>	The label of the structure containing the list of controls to include in the HTML layout and generation process.
<i>html object</i>	The label of the HtmlClass object that writes the HTML code.
<i>style</i>	A string constant, variable, EQUATE, or expression containing any HTML STYLE specifications to apply to all the controls.
<i>snapX</i>	An integer constant, variable, EQUATE, or expression containing a horizontal scaling factor used in the layout process. Typically, the WebWindowClass.SnapX property.
<i>snapY</i>	An integer constant, variable, EQUATE, or expression containing a vertical scaling factor used in the layout process. Typically, the WebWindowClass.SnapY property.

The **CreateHtml** method generates HTML code to represent each visible control in the *control list*. The HTML code generation includes intelligent layout of the controls on the resulting Web page.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

### Implementation:

Typically, the WebWindowClass.GetChildren method builds the *control list* prior to the call to CreateHtml. CreateHtml instantiates a LayoutHtmlClass to arrange the controls and calls the AddControlsToLayout method to include only the appropriate controls in the layout process.

The *control list* parameter must name a QUEUE with a structure the same as the WebControlQueue declared in ICWINDOW.INC:

```
WebControlQueue  QUEUE,TYPE
Feq              SIGNED
ThisControl      &WebControlClass
END
```

Example:

```
WebWindowClass.CreateChildHtml PROCEDURE|  
    (*HtmlClass Target, SIGNED ParentFq, STRING Style)
```

```
Controls                WebControlQueue
```

```
CODE  
SELF.GetChildren(Controls, ParentFq)  
SELF.CreateHtml(Controls, Target, Style, SELF.SnapX, SELF.SnapY)
```

See Also: **AddControlsToLayout, WebWindowClass.GetChildren,  
WebWindowClass.SnapX, WebWindowClass.SnapY, LayoutHtmlClass**

## Init (initialize the WebControlListClass object)

---

### Init, VIRTUAL

The **Init** method is only a virtual placeholder method to initialize the WebControlListClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also:           WebWindowClass.Init

## Kill (shut down the WebControlListClass object)

---

### Kill, VIRTUAL

The **Kill** method is only a virtual placeholder method to shut down the WebControlListClass object.

Kill is a VIRTUAL method so that other base class methods can directly call the Kill virtual method in a derived class. This lets you easily implement your own custom version of this method.

## SetParentDefaults (set a control's children)

**SetParentDefaults**( *control list*, *parent*, *coordinates* )

**SetParentDefaults** Identifies all the children of a *parent*.

*control list*            The label of a structure containing a list of controls.

*parent*                    The label of the WebControlClass object whose children are identified.

*coordinates*            The label of a structure containing the positional coordinates of the *parent*.

The **SetParentDefaults** method identifies each child of the *parent*. A child is any control in the *control list* whose coordinates fall entirely within *coordinates*. This positional determination of parent/child relationships results in proper alignment and processing of controls on the Web page.

Implementation:

The ConfirmParent method identifies each child of the *parent* by setting the child's ParentFeq property equal to the *parent*'s Feq property. The SetParentDefaults method calls the WebControlClass.SetParentDefaults method for each control in the *control list*.

The *control list* parameter must name a QUEUE with a structure the same as the WebControlQueue QUEUE declared in ICWINDOW.INC:

```
WebControlQueue  QUEUE,TYPE
Feq              SIGNED
ThisControl      &WebControlClass
END
```

The *coordinates* parameter names a GROUP with a structure the same as the Rect GROUP declared in ICWINDOW.INC:

```
Rect            GROUP,TYPE
x              SIGNED
y              SIGNED
width          SIGNED
height         SIGNED
END
```

Example:

```
WebHtmlImageClass.SetChildDefaults  PROCEDURE
```

```
Children          WebControlQueue
MyRect            GROUP(Rect)
END
```

```
CODE
SELF.GetPosition(MyRect.x, MyRect.y, MyRect.width, MyRect.height)
SELF.OwnerWindow.GetChildren(Children, SELF.ParentFeq, CREATE:Region)
SELF.OwnerWindow.SetParentDefaults(Children, SELF, MyRect)
```

See Also:

**WebControlClass.SetParentDefaults**

## WebWindowBaseClass Properties

The WebWindowBaseClass contains the properties listed below.

### AllowJava (generate or suppress JavaScript)

---

**AllowJava**      **BYTE**

The **AllowJava** property indicates whether to generate any JavaScript within the generated HTML code. A value of one (1) allows generation of JavaScript; a value of zero (0) suppresses generation of JavaScript.

**Tip:**      **It is a good idea to suppress JavaScript for splash screens because the combination of Java applets and the Clarion TIMER attribute can trigger bugs in some browsers, causing them to lock up.**

Implementation:      You can use the AllowJava property to suppress all JavaScript throughout your Web-enabled application to produce a limited function program that is accessible from non-Java browsers.

The Init method sets the initial value of the AllowJava property to TRUE.

See Also:              Init

### Background (window background color)

---

**Background**      **LONG(COLOR:None)**

The **Background** property indicates the background color of the Web page window. Discrete areas of the window (caption, menubar, toolbar, and client area) inherit this color unless overridden.

Implementation:      The SetBackground method sets the value of the Background property.

See Also:              SetBackground

### BackImage (window wallpaper)

---

**BackImage**      **ANY**

The **BackImage** property indicates the background image (wallpaper) to apply to the Web page window. Discrete areas of the window (caption, menubar, toolbar, and client area) inherit this image unless overridden.

Implementation:      The SetBackground method sets the value of the BackImage property.

See Also:              SetBackground

## BorderWidth (Web page border width)

---

**BorderWidth**    **BYTE**

The **BorderWidth** property indicates the width or thickness of the Web page border. A value of zero (0) displays no border. Larger values result in wider borders.

Implementation:    The Init method sets the initial value of the BorderWidth property to two (2).

See Also:    Init

## CloseImage (close button graphic)

---

**CloseImage**    **STRING(FILE:MaxFileName)**

The **CloseImage** property contains the pathname of the image to display on the Close Button generated by the WebWindowClass object.

Implementation:    The WebWindowClass object creates the Close Button (or not) based on the value of the CreateClose property. The Init method sets the value of the CloseImage property. The default CloseImage value is 'exit.ico.'

See Also:    CreateClose, Init

## CreateCaption (include a titlebar on the Web page)

---

**CreateCaption**    **BYTE**

The **CreateCaption** property indicates whether to include a caption (titlebar) on the Web page. A value of one (1) includes a caption; a value of zero (0) omits the caption.

**Tip:**    **The WINDOW's title text always displays in the browser's titlebar. Therefore, including a caption on your Web page does not supply any additional information to the end user; however, it does preserve a consistent appearance between the application when running under Windows and when running under a browser.**

Implementation:    The Init method sets the initial value of the CreateCaption property to TRUE. The ResetFromControls method creates the caption.

See Also:    Init, ResetFromControls

## CreateClose (include a close button on the Web page)

---

### CreateClose BYTE

The **CreateClose** property indicates under what circumstances to include a close button on the Web page. Circumstances include always, never, when the WINDOW has a system menu (the SYSTEM attribute), and when the WINDOW has a system menu and no “visible” BUTTONs.

The WebWindowClass provides this automatic close button primarily as a substitute for the system menu because system menus are not supported by HTML.

Implementation: EQUATEs for the CreateClose property are declared in ICWINDOW.INC as follows:

```

ITEMIZE(0),PRE(CLOSE)
Never      EQUATE
IfSystem   EQUATE
SystemNoButton EQUATE
Always     EQUATE
END

```

The Init method sets the initial value of the CreateClose property to CLOSE:Always. The ResetFromControls method creates the close button.

The CloseImage property specifies the image to display on the close button.

See Also: CloseImage, Init, ResetFromControls

## CreateToolbar (include a toolbar on the Web page)

---

### CreateToolbar BYTE

The **CreateToolbar** property indicates whether to include a toolbar on the Web page. A value of one (1) includes a toolbar if there is at least one visible toolbar control; a value of zero (0) omits the toolbar even if the window contains visible toolbar controls.

Implementation: The Init method sets the CreateToolbar property to TRUE. The ResetFromControls method creates the toolbar.

See Also: Init, ResetFromControls

## DefaultButton (enter key button)

---

### DefaultButton SIGNED

The **DefaultButton** property contains the control number of the WINDOW's default button (DEFAULT attribute). The WebWindowClass uses this property to invoke the button's associated action when the end-user presses the ENTER key.

## DefaultButtonNeeded (simulate default button)

---

### DefaultButtonNeeded BYTE

The **DefaultButtonNeeded** property indicates whether the WebWindowClass object must simulate a default button on the Web page where no default button exists. This property helps the WebWindowClass object synchronize the behavior of the ENTER key when running the Server application under Windows versus running it under a Web browser.

Under Standard Windows Behavior, pressing the ENTER key when there is no default button does nothing. Under most browsers pressing the ENTER key when there is no default button invokes an action roughly equivalent to the typical Windows OK button (submits data from the current Web page and optionally requests a new page). The DefaultButtonNeeded property allows the WebWindowClass object to accurately simulate a default button on a Web page that doesn't actually have one.

## DisabledAction (default HTML for disabled controls)

---

### DisabledAction BYTE

The **DisabledAction** property sets the default treatment of any disabled controls for generating HTML to represent them. This property lets you determine how to handle controls that are disabled under Windows, but cannot be disabled under a browser, because the HTML control does not support disabling.

Valid treatments include, always display the disabled control, always hide the disabled control, and disable the control if HTML supports it, otherwise hide the control.

**Tip:** The `WebControlClass.DisabledAction` property overrides the `WebWindowClass.DisabledAction` property for any individual control.

Implementation: EQUATEs for the DisabledAction property are declared in ICWINDOW.INC as follows:

```

ITEMIZE,PRE(DISABLE)
Hide      EQUATE      !always hide disabled control
OptHide   EQUATE      !hide if HTML control won't disable
Show      EQUATE      !always show disabled control
END

```

The Init method sets the DisabledAction property to DISABLE:Hide.

See Also: Init, WebControlClass.DisabledAction

## Files (WebClientManagerClass object)

---

### Files &WebFilesClass

The **Files** property is a reference to the WebFilesClass object that manages pathnames for the WebWindowClass object. The WebWindowClass uses this property to generate appropriate pathnames wherever needed.

Implementation: The Init method sets the Files property to reference the Server.Files property.

See Also: Init, WebServerClass.Files

## FormatBorderWidth (HTML table cell border width)

---

### FormatBorderWidth    BYTE

The **FormatBorderWidth** property indicates the width or thickness of each HTML table cell in the Web page. A value of zero (0) displays no borders. Larger values result in wider borders.

This property is primarily a debugging tool to help you see how and why each control is placed where it is on the generated Web page.

Implementation:        The Init method sets the FormatBorderWidth property to zero (0).

See Also:                Init

## GroupBorderWidth (group box border width)

---

### GroupBorderWidth    BYTE

The **GroupBorderWidth** property indicates the width or thickness of Web page Group box borders. A value of zero (0) displays no border. Larger values result in wider borders.

Implementation:        The Init method sets the GroupBorderWidth property to two (2).

See Also:                Init

## HelpDocument (HTML help document)

---

### HelpDocument    ANY

The **HelpDocument** property contains the pathname or the URL of the HTML help document for the Server (Web-enabled application).

Implementation:        The SetHelpDocument method or the SetHelpURL method sets the value of the HelpDocument property.

See Also:                SetHelpDocument, SetHelpURL

## HelpEnabled (HTML help enabled flag)

---

HelpEnabled	BYTE
-------------	------

The **HelpEnabled** property indicates whether a help document is specified for the WebWindowClass object. A value of one (1 or True) indicates a help document; a value of zero (0 or False) indicates no help document.

Implementation: The SetHelpDocument method or the SetHelpURL method sets the value of the HelpDocument property to True.

See Also: HelpDocument, SetHelpDocument, SetHelpURL

## HelpRelative (remote or local help document)

---

HelpRelative	BYTE
--------------	------

The **HelpRelative** property indicates whether the HelpDocument property contains a pathname relative to the Application Broker or an independent URL. A value of one (1 or True) indicates a pathname relative to the Application Broker; a value of zero (0 or False) indicates an independent URL.

Implementation: The SetHelpDocument method sets the HelpRelative property to True. The SetHelpURL method sets the HelpRelative property to False.

See Also: HelpDocument, SetHelpDocument, SetHelpURL

## HelpStyle (HTML help style)

---

HelpStyle	ANY
-----------	-----

The **HelpStyle** property contains...

Implementation: The SetHelpDocument and SetHelpURL methods set the HelpStyle property.

See Also: SetHelpDocument, SetHelpURL

## HtmlOption (window/control scaling information)

---

### HtmlOption      LIKE(HtmlOptionGroup)

The **HtmlOption** property contains information for proper scaling and positioning of controls on the Web page. The actual effect of this property is determined by the `HtmlClass` object that applies the information.

Implementation:

The `SetFormatOptions` method sets the values of the `HtmlOption` property. These values are used for converting dialog units to pixels for scaling purposes.

The `HtmlOptionGroup` is declared in `ICHTML.INC` as follows:

```
HtmlOptionGroup GROUP,TYPE
ScaleX           REAL      !horizontal conversion factor
ScaleY           REAL      !vertical conversion factor
END
```

The `HtmlClass` object stores `HtmlOption` information in the `HtmlClass.Option` property.

See Also:

`SetFormatOptions`, `HtmlClass.Option`

## IsSplash (splash screen flag)

---

### IsSplash      BYTE

The **IsSplash** property indicates whether the `WINDOW` serves as a splash screen. A value of one (1) indicates a splash screen; a value of zero (0) indicates some other purpose. The `WebWindowClass` uses this property to generate any HTML unique to splash screens.

Implementation:

The `SetSplash` method sets the value of the `IsSplash` property.

By convention, Windows splash screens close immediately upon an end user mouse-click anywhere in the window. Browsers and HTML do not support this behavior; therefore, the `WebWindowClass` generates an HTML control to accept the mouse-click and close the window.

See Also:

`SetSplash`

## MenubarFeq (menubar control number)

---

**MenubarFeq**    **SIGNED**

The **MenubarFeq** property contains the control number (field equate) of the MENUBAR. The WebWindowClass uses this property to refer to the WINDOW's MENUBAR structure.

Implementation:    The Init method sets the initial value of the MenubarFeq property. The GetMenubarFeq method returns the value of the MenubarFeq property.

See Also:    Init, GetMenubarFeq

## MenubarType (menu placement)

---

**MenubarType**    **SIGNED**

The **MenubarType** property indicates where the WebWindowClass places the menu items on the Web page. Valid options are above the toolbar, left side of the window, or omit the menu items from the Web page.

Implementation:    The Init method sets the MenubarType property to PROP:none which omits menu items. To include menu items on the Web page, you must assign a value to the MenubarType property after the Init method executes.

EQUATEs for the MenubarType property are declared in ICWINDOW.INC as follows:

```
PROP:none    EQUATE(0)
```

and in PROPERTY.CLW as follows:

```
PROP:above    EQUATE(7C0AH) ! 0 = off, else on
PROP:left    EQUATE(7C08H) ! 0 = off, else on
```

See Also:    Init

## OptionBorderWidth (option box border width)

---

**OptionBorderWidth**    **BYTE**

The **OptionBorderWidth** property indicates the width or thickness of Web page Option box borders. A value of zero (0) displays no border. Larger values result in wider borders.

Implementation:    The Init method sets the OptionBorderWidth property to two (2).

See Also:    Init

## PageBackground (web page background color)

---

**PageBackground**      **LONG(COLOR:None)**

The **PageBackground** property indicates the background color of the Web page.

Implementation:      The SetPageBackground method sets the value of the PageBackground property.

See Also:              SetPageBackground

## PageImage (web page wallpaper)

---

**PageImage**      **ANY**

The **PageImage** property indicates the background image (wallpaper) to apply to the Web page.

Implementation:      The SetPageBackground method sets the value of the PageImage property.

See Also:              SetPageBackground

## Server (WebServerClass object)

---

**Server**      **&WebServerClass**

The **Server** property is a reference to the WebServerClass object that represents the Server application for the WebWindowClass object. The WebWindowClass relies on this property to supply information about the Client browser, to supply appropriate pathnames, to handle events originating from the Client browser, and to interact with the BrokerClass object as needed.

Implementation:      The Init method sets the initial value of the Server property—typically to reference a global instance of the WebServerClass.

See Also:              Init

## SheetBorderWidth (sheet border width)

---

### SheetBorderWidth BYTE

The **SheetBorderWidth** property indicates the width or thickness of Web page Sheet borders. A value of zero (0) displays no border. Larger values result in wider borders.

Implementation: The Init method sets the SheetBorderWidth property to two (2).

See Also: Init

## SnapX (horizontal control alignment factor)

---

### SnapX SIGNED

The **SnapX** property contains a horizontal “snap to” factor for aligning or laying out controls on the Web page. The actual effect of the this property is determined by the LayoutHtmlClass object that arranges the window controls and applies the factor.

Implementation: The SetFormatOptions method sets the initial value of the SnapX property.

See Also: SetFormatOptions, LayoutHtmlClass.Init

## SnapY (vertical control alignment factor)

---

### SnapY SIGNED

The **SnapY** property contains a vertical “snap to” factor for aligning or laying out controls on the Web page. The actual effect of the this property is determined by the LayoutHtmlClass object that arranges the window controls and applies the factor.

Implementation: The SetFormatOptions method sets the initial value of the SnapY property.

See Also: SetFormatOptions, LayoutHtmlClass.Init

## TimerAction (time released browser action)

---

**TimerAction**     **BYTE, PROTECTED**

The **TimerAction** property specifies an action the browser takes after the time interval set by the **TimerDelay** property. The action is repeated each time the **TimerDelay** interval expires.

There are four valid actions, three of which consist of a request to the Server for an update of the Web page. **Update:Full** submits any entered items and requests a complete screen redraw. **Update:Partial** submits any entered items and requests only the data to fill Java controls (see *JSL Manager Class*). **Update:Refresh** is only valid for the windows with a **TimerDelay**. It submits no items but requests a complete screen redraw when the timer runs out. **Update:OnBrowser** updates the Web page as far as possible without contacting the Server.

This property is **PROTECTED**, therefore, it can only be referenced by a **WebWindowClass** method, or a method in a class derived from **WebWindowClass**.

Implementation:     EQUATEs for the **TimerAction** property are declared in **ICSTD.INC** as follows:

```
Update:OnBrowser     EQUATE(0)
Update:Partial       EQUATE(1)
Update:Full          EQUATE(2)
Update:Refresh       EQUATE(3)
```

The **Init** method sets the initial value of the **TimerAction** property to zero (0). The **SetTimer** method sets subsequent values of the **TimerAction** property.

See Also:             **Init, SetSplash, SetTimer, TimerDelay**

## TimerDelay (time interval for browser action)

---

**TimerDelay**     **SIGNED, PROTECTED**

The **TimerDelay** property contains the time period for Java applets to wait before initiating the action specified by the **TimerAction** property.

This property is **PROTECTED**, therefore, it can only be referenced by a **WebWindowClass** method, or a method in a class derived from **WebWindowClass**.

Implementation:     The **Init** method sets the initial value of the **TimerDelay** property to zero (0). The **SetTimer** method sets subsequent values of the **TimerDelay** property.

See Also:             **Init, SetSplash, SetTimer, TimerAction**

## ToolbarFeq (toolbar control number)

---

ToolbarFeq	SIGNED
------------	--------

The **ToolbarFeq** property contains the control number (field equate) of the TOOLBAR. The WebWindowClass uses this property to refer to the WINDOW's TOOLBAR structure.

Implementation: The Init method sets the initial value of the ToolbarFeq property. The GetToolbarFeq method returns the value of the ToolbarFeq property.

See Also: Init, GetToolbarFeq

## WebWindowBaseClass Methods

The WebWindowBaseClass inherits all the methods of the WebControlListClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebWindowBaseClass contains the methods listed below.

### Functional Organization—Expected Use

---

The WebWindowBaseClass is an abstract class. Its methods are not implemented; rather, they are simply placeholders for derived classes such as the WebWindowClass. See *WebWindowClass Methods—Functional Organization—Expected Use* for more information.

### CreateChildHtml (create HTML for control's children)

---

**CreateChildHtml( *html target*, *parent control* [, *border width*] ), VIRTUAL**

**CreateChildHtml** A virtual placeholder to generate HTML code for each child control of the parent control.

*html target* The label of the HtmlClass object that writes the HTML code.

*parent control* An integer constant, variable, EQUATE, or expression containing parent control's the field equate number, or the WINDOW's field equate number, that is, zero (0).

*border width* An integer constant, variable, EQUATE, or expression containing the border width for the parent control. If omitted, border width defaults to zero (0).

The **CreateChildHtml** method is only a virtual placeholder to generate HTML code for each child control of the parent control.

CreateChildHtml is a VIRTUAL method so that other base class methods can directly call the CreateChildHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also:

WebWindowClass.CreateChildHtml

## GetBackgroundColor (return Web window background color)

---

**GetBackgroundColor**( [*default color*] ), LONG, VIRTUAL

### **GetBackgroundColor**

A virtual placeholder to return the background color of the Web page window.

*default color* An integer variable, constant, EQUATE, or expression containing the color to return if there is no background color. If omitted, *default color* defaults to Color:None.

The **GetBackgroundColor** method is only a virtual placeholder to return the background color of the Web page.

GetBackgroundColor is a VIRTUAL method so that other base class methods can directly call the GetBackgroundColor virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: LONG

See Also: Background, WebWindowClass.GetBackgroundColor

## GetBackgroundImage (return Web window wallpaper)

---

**GetBackgroundImage**, STRING, VIRTUAL

The **GetBackgroundImage** method is only a virtual placeholder to return the filename containing the background image of the Web page window.

GetBackgroundImage is a VIRTUAL method so that other base class methods can directly call the GetBackgroundImage virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: STRING

See Also: BackImage, WebWindowClass.GetBackgroundImage

## GetControl (return control information)

---

**GetControl( control [, control information] ), BYTE, VIRTUAL**

<b>GetControl</b>	A virtual placeholder to return information for the specified <i>control</i> .
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control's field equate number.
<i>control information</i>	The label of a structure to contain the control information. If omitted, GetControl returns a true or false value indicating whether the WebWindowClass object "knows" about the <i>control</i> .

The **GetControl** method is only a virtual placeholder to return information for the specified *control*.

GetControl is a VIRTUAL method so that other base class methods can directly call the GetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The *control information* parameter names a structure like the WebControlRefGroup GROUP declared in ICWINDOW.INC as follows:

```
WebControlRefGroup  GROUP,TYPE
Control              &WebControlClass
                    END
```

Return Data Type:

BYTE

See Also:

WebWindowClass.GetControl

## GetCreateClose (return close button flag)

---

**GetCreateClose, BYTE, VIRTUAL**

The **GetCreateClose** method is only a virtual placeholder. It returns a value indicating whether to create a close button on the Web page.

GetCreateClose is a VIRTUAL method so that other base class methods can directly call the GetCreateClose virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type:

BYTE

See Also:

WebWindowClass.GetCreateClose

## GetChildren (return all child controls)

**GetChildren**( *children*, *parent control* [, *control type*] ), **VIRTUAL**

<b>GetChildren</b>	A virtual placeholder to return information about the children of the specified parent control.
<i>children</i>	The label of a structure to contain the references or control numbers of the child controls.
<i>parent control</i>	An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.
<i>control type</i>	An integer constant, variable, EQUATE, or expression indicating the type of control to include, or exclude. A negative control type excludes controls of the specified type. A positive control type includes controls of the specified type. If omitted, all control types are included.

The **GetChildren** method is only a virtual placeholder to return information about the children of the specified parent control.

GetChildren is a VIRTUAL method so that other base class methods can directly call the GetChildren virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The *children* parameter is a QUEUE with the same structure as the WebControlQueue declared in ICWINDOW.INC as follows:

```
WebControlQueue  QUEUE,TYPE
Feq              SIGNED
ThisControl      &WebControlClass
END
```

EQUATEs for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE:.

See Also:

WebWindowClass.GetChildren

## GetFirstChild (return first child control)

---

**GetFirstChild**( *parent control* [, *control type*] ), **SIGNED, VIRTUAL**

**GetFirstChild** A virtual placeholder to return the control number of the first visible child of the specified parent control.

*parent control* An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.

*control type* An integer constant, variable, EQUATE, or expression containing the type of child control for which to search. If omitted, any control type is valid.

The **GetFirstChild** method is only a virtual placeholder to return the control number of the first visible child of the specified parent control.

GetFirstChild is a VIRTUAL method so that other base class methods can directly call the GetFirstChild virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: EQUATES for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE:.

Return Data Type: SIGNED

See Also: WebWindowClass.GetFirstChild

## GetHelpHandler (return HTML to show help document)

---

**GetHelpHandler**, **STRING, VIRTUAL**

The **GetHelpHandler** method is only a virtual placeholder to return the HTML to request the help document associated with the Web page.

GetHelpHandler is a VIRTUAL method so that other base class methods can directly call the GetHelpHandler virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: STRING

See Also: HelpDocument, WebWindowClass.GetHelpHandler

## GetHelpReference (return HTML help document reference)

---

### GetHelpReference, STRING, VIRTUAL

The **GetHelpReference** method is only a virtual placeholder to return the HTML reference for a specific help display. The returned value depends on the values of the HelpDocument and HelpRelative properties.

GetHelpReference is a VIRTUAL method so that other base class methods can directly call the GetHelpReference virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: STRING

See Also: HelpDocument, HelpRelative, WebWindowClass.GetHelpReference

## GetHelpTarget (return HTML HREF for help display)

---

### GetHelpTarget, STRING, VIRTUAL

The **GetHelpTarget** method is only a virtual placeholder to return the HTML <HREF> for a specific help display. The returned value depends on the values of the HelpDocument and HelpRelative properties.

GetHelpTarget is a VIRTUAL method so that other base class methods can directly call the GetHelpTarget virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: STRING

See Also: GetHelpReference, HelpDocument, HelpRelative, WebWindowClass.GetHelpTarget

## GetMenubarFeq (return menubar control number)

---

### GetMenubarFeq, SIGNED, VIRTUAL

The **GetMenubarFeq** method is only a virtual placeholder to return the control number (field equate) of the window's MENUBAR.

GetMenubarFeq is a VIRTUAL method so that other base class methods can directly call the GetMenubarFeq virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: SIGNED

See Also: MenubarFeq, WebWindowClass.GetMenubarFeq

## GetPageImage (return Web page wallpaper)

---

### GetPageImage, STRING, VIRTUAL

The **GetPageImage** method is only a virtual placeholder to return the filename of the file containing the background image of the Web page.

GetPageImage is a VIRTUAL method so that other base class methods can directly call the GetPageImage virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type:      STRING

See Also:                PageImage, WebWindowClass.GetPageImage

## GetShowMenubar (return menubar include/omit flag)

---

### GetShowMenubar, BYTE, VIRTUAL

The **GetShowMenubar** method is only a virtual placeholder to return a value indicating whether to include a menubar on the window.

GetShowMenubar is a VIRTUAL method so that other base class methods can directly call the GetShowMenubar virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type:      BYTE

See Also:                WebWindowClass.GetShowMenubar

## GetShowToolbar (return toolbar include/omit flag)

---

### GetShowToolbar, BYTE, VIRTUAL

The **GetShowToolbar** method is only a virtual placeholder to return a value indicating whether to include a toolbar on the window.

GetShowToolbar is a VIRTUAL method so that other base class methods can directly call the GetShowToolbar virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type:      BYTE

See Also:                WebWindowClass.GetShowToolbar

## GetToolBarFeq (return Toolbar control number)

---

### GetToolBarFeq, SIGNED, VIRTUAL

The **GetToolBarFeq** method is only a virtual placeholder to return the control number (field equate) of the window's TOOLBAR.

GetToolBarFeq is a VIRTUAL method so that other base class methods can directly call the GetToolBarFeq virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: SIGNED

See Also: ToolBarFeq, WebWindowClass.GetToolBarFeq

## GetToolBarMode (return toolbar entity)

---

### GetToolBarMode, BYTE, VIRTUAL

The **GetToolBarMode** method is only a virtual placeholder to return a value indicating what the toolbar is driving. For example, a BrowseBox, a Relation Tree, or an update form.

GetToolBarMode is a VIRTUAL method so that other base class methods can directly call the GetToolBarMode virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: EQUATEs for GetToolBarMode return values are declared in TPLEQU.CLW as follows:

FormMode	EQUATE(1)
BrowseMode	EQUATE(2)
TreeMode	EQUATE(3)

Return Data Type: BYTE

See Also: WebWindowClass.GetToolBarMode

## GetWebActiveFrame (return WebFrameClass reference)

---

### GetWebActiveFrame, WebFrameClass, VIRTUAL

The **GetWebActiveFrame** method is only a virtual placeholder to return a reference to the active **WebFrameClass**.

**GetWebActiveFrame** is a **VIRTUAL** method so that other base class methods can directly call the **GetWebActiveFrame** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: **WebFrameClass**

See Also: **WebWindowClass.GetWebActiveFrame**

## WebWindowClass Properties

The WebWindowClass inherits all the properties of the WebWindowBaseClass from which it is derived. See *WebWindowBaseClass Properties* for more information.

In addition to the inherited properties, the WebWindowClass contains the properties listed below.

### Authorize (require username and password)

---

Authorize	BYTE
	<p>The <b>Authorize</b> property indicates whether a valid username and password are required to access the Web page representing this window.</p> <p>A value of one (1) indicates username and password are required. A value of zero (0) indicates no username and password are required. The WebWindowClass uses this property to provide procedure level security.</p>
Implementation:	<p>The Init method sets the initial value of the Authorize property to False. The SetPassword method sets the Authorize property to True.</p> <p>The SetPassword method sets the Authorize property to true, to password protect the Web page (AuthorizeArea). It also sets the authorized password for the AuthorizeArea. The GetAuthorized method collects and validates the end user's password for the AuthorizeArea. The ValidatePassword method verifies the password entered by the end user. Finally, the TakeCreatePage method generates an "access denied" page (CreateUnauthorizedPage) for invalid passwords.</p>
See Also:	AuthorizeArea, CreateUnauthorizedPage, GetAuthorized, TakeCreatePage, SetPassword, ValidatePassword

## AuthorizeArea (name of password protected Web page)

---

### AuthorizeArea ANY

The **AuthorizeArea** property contains the identity of a Web page (procedure) within the Server application that may be password protected. The Client browser displays the contents of the AuthorizeArea property when prompting the end user for an authorized username and password.

#### Implementation:

The Init method sets the initial value of the AuthorizeArea. The Internet Connect Templates set AuthorizeArea to *window title + procedure name*.

The SetPassword method sets the Authorize property to true, to password protect the Web page (AuthorizeArea). It also sets the authorized password for the AuthorizeArea. The GetAuthorized method collects and validates the end user's password for the AuthorizeArea. The ValidatePassword method verifies the password entered by the end user. Finally, the TakeCreatePage method generates an "access denied" page (CreateUnauthorizedPage) for invalid passwords.

#### See Also:

Authorize, CreateUnauthorizedPage, GetAuthorized, TakeCreatePage, SetPassword, ValidatePassword

## HtmlTarget (HtmlClass object)

---

### HtmlTarget &HtmlClass

The **HtmlTarget** property is a reference to the HtmlClass object that writes HTML code representing the window. The WebWindowClass uses this property to generate HTML code.

#### Implementation:

The Init method sets the value of the HtmlTarget property.

#### See Also:

Init

## IsCentered (center or left-justify web window)

---

### IsCentered BYTE

The **IsCentered** property indicates whether the window is centered within the generated Web page. A value of one (1) centers the window; a value of zero (0) left justifies the window.

#### Implementation:

The Init method sets the value of the IsCentered property to True.

#### See Also:

Init

## IsSecure (public or secure channel)

---

**IsSecure****BYTE**

The **IsSecure** property determines whether HTML code and JSL data representing the window are sent through a secure channel or a public channel. A value of one (1 or True) transmits on a secure channel; a value of zero (0 or False) transmits on a public channel.

There are significant performance penalties associated with the use of secure channels, so we recommend using them only when circumstances demand it; for example, when transmitting credit card numbers or other very sensitive information.

Secure channels are not supported by all configurations of all Web-servers. See *Security in the Application Broker* in the *Interconnect Connect User's Guide* for more information on setting up secure channels.

**Implementation:**

The Init method sets the IsSecure property to False.

A secure channel consists of a secure Web-server directory to contain any intermediate or temporary files, plus encryption of data prior to its transmission, and finally, decryption of the data when it reaches its destination.

**See Also:**

Init

## SentHtml (first time process)

---

**SentHtml****BYTE**

The **SentHtml** property indicates whether the HTML code representing the window has been sent to the Client at least once. A value of one (1) indicates the page was sent; a value of zero (0) indicates the page has not been sent.

The WebWindowClass uses this property to ensure a page is sent to the browser before any updates to the page are sent (that is, when you change procedures/windows), and to initiate any one-time only processing for the window, such as prompting for Username and Password.

**Implementation:**

The CreateHtmlPage method sets the SentHtml property to True.

**See Also:**

CreateHtmlPage

## WebWindowClass Methods

The WebWindowClass inherits all the methods of the WebWindowBaseClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebWindowClass contains the methods listed below.

### Functional Organization—Expected Use

---

As an aid to understanding the WebWindowClass, it is useful to organize its methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### Primary Interface Methods

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize WebWindowClass object
SetBackground	set Web window background
SetPageBackground	set Web page background
SetFormatOptions	set Web page scale and snap to
SetHelpDocument	enable Web page help
SetHelpURL	enable Web page help
SetParentDefaults	set default parent for each control
SetPassword	require password
SetSplash	make this a splash window
SetTimer	set Web page timer and action
Kill <sup>v</sup>	shut down WebWindowClass object

<sup>v</sup> These methods are also Virtual.

##### **Mainstream Use:**

TakeEvent	handle browser & ACCEPT loop events
-----------	-------------------------------------

##### **Occasional Use:**

CreateDummyHtmlPage	write empty Html page
CreateHtmlPage	generate HTML for a window
CreateJsldata	generate Java Support Library data
CreatePageFooter	generate HTML page footer
CreatePageHeader	generate HTML page footer
CreateUnauthorizedPage	create unauthorized user page
GetAuthorized	check user authorization
GetButtonInClientArea	return button present indicator

GetHelpHandler	return HTML to show help document
GetHelpReference	return help document reference
GetHelpTarget	return HTML HREF for help display
GetTargetSecurity	return public or secure flag
SetCentered	center or left-justify window
SuppressControl	omit control from Web page
ResetFromControls	set control information

### **Virtual Methods**

Typically you will not call these methods directly—the Primary Interface methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

AddControl	add control information
AddControlsToLayout	set controls for HTML generation
BodyFooter	generate HTML BODY footer
BodyHeader	generate HTML BODY header
CreateChildHtml	create HTML for control's children
CreateHtml	generate HTML for controls
GetBackgroundColor	return Web window background color
GetBackgroundImage	return Web window wallpaper
GetChildren	return all child controls
GetControl	return control information
GetControlInfo	return control reference
GetCreateClose	return close button flag
GetFirstChild	return first child control
GetHelpHandler	return HTML to show help document
GetHelpReference	return HTML help reference
GetHelpTarget	return HTML HREF for help display
GetMenubarFeq	return MENUBAR field equate
GetPageImage	return Web page wallpaper
GetShowMenubar	return menubar include/omit flag
GetShowToolbar	return toolbar include/omit flag
GetToolbarFeq	return TOOLBAR field equate
GetToolbarMode	return toolbar entity
GetWebActiveFrame	return active WebFrameClass reference
Kill	shut down WebWindowClass object
TakeCreatePage	fill Client request for page
TakeRequest	process browser event/request
TakeUnknownSubmit	process unknown field assignment
TitleContents	set browser titlebar
ValidatePassword	verify password

## AddControl (add control information)

**AddControl( *control* [,*type*] ), &WebControlClass, VIRTUAL, PROC**

<b>AddControl</b>	Adds information about a control or pseudo-control for this window.
<i>control</i>	An integer constant, variable, EQUATE, or expression containing a control number or the label of a WebControlClass object.
<i>type</i>	An integer constant, variable, EQUATE, or expression indicating the type of control or the label of a WebControlClass object. If omitted, AddControl intelligently determines the appropriate control type.

The **AddControl** method adds information about a control or pseudo-control for this window. The WebWindowClass object uses this information to generate HTML code to represent the window in the Client browser.

The information may represent a Clarion control within the WINDOW structure, a pseudo-control, such as the window titlebar or client area, or it may represent a control that appears only on the Web page and not on the WINDOW, such as a global toolbar button on an MDI child window.

AddControl is a VIRTUAL method so that other base class methods can directly call the AddControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method has the PROC attribute, so you can call it like a PROCEDURE and ignore the return value.

### AddControl(*control*)

If *control* is a field equate, AddControl instantiates a WebControlClass object of the appropriate type, adds it to the control list, and returns a reference to the new control object.

If *control* is a WebControlClass object, AddControl adds the object to the control list, but does not initialize it. There is no return value.

### AddControl(*control*, *type*)

If *type* is a control type, AddControl instantiates a WebControlClass object of the specified type, then adds it to the control list, and returns a reference to the new control object.

If *type* is a WebControlClass object, AddControl initializes the object, then adds it to the control list. There is no return value.

Return Data Type: &WebControlClass

Implementation: The AddControl method adds a WebControlClass object to the list of control

objects for this window. The `AddControl` method may instantiate a *new* `WebControlClass` object then add it to the list, or it may add an *existing* object to the list.

Example:

```
WebMenubar WebMenubarClass                                !declare WebMenubar control object
CODE
WebWindow.AddControl(?InsertButton)                       !instantiate & add a real control
IC:CurControl &= WebWindow.AddControl(?CustList)!instantiate & add a real control
IC:CurControl.SetQueue(CustQ)                            ! and use the returned reference
WebWindow.AddControl(?Feq:Menubar, WebMenubar)           !add an existing control object
SELF.AddControl(FEQ:Caption, CREATE:Caption)              !instantiate & add a psuedo-control
SELF.AddControl(FEQ:Close, CREATE:Close)                  !instantiate & add Web only control
```

See Also: [WebControlClass](#)

## AddControlsToLayout (set controls for HTML generation)

**AddControlsToLayout**( *layout object*, *parent control* ), VIRTUAL

### AddControlsToLayout

Sets the controls to include in the HTML layout and generation process.

*layout object* The label of the LayoutHtmlClass object that optimally arranges the controls on the generated Web page.

*parent control* An integer constant, variable, EQUATE, or expression containing a control number of the control whose children are included in the layout process, or zero (0) if the window is the parent.

The **AddControlsToLayout** method sets the controls to include in the HTML layout and generation process. The AddControlsToLayout method includes only “visible” controls; that is, it excludes disabled and hidden controls from the Web page layout process.

AddControlsToLayout is a VIRTUAL method so that other base class methods can directly call the AddControlsToLayout virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: AddControlsToLayout selects from the WebWindowClass’ list of controls. It assumes the control list has already been built by appropriate calls to the AddControl method, and includes only the “visible” controls from this list into the HTML layout process. The GetVisible method evaluates whether a control is visible.

Example:

```
WebControlListClass.CreateHtml PROCEDURE(*WebControlQueue Source, |
    *HtmlClass Target, STRING style, SIGNED SnapX, SIGNED SnapY)
```

```
Layout                LayoutHtmlClass

CODE
Layout.Init(style, SnapX, SnapY)
SELF.AddControlsToLayout(Source, Layout)
Layout.CreateHtml(Target)
Layout.Kill
```

See Also: AddControl, GetVisible

## BodyFooter (generate HTML BODY footer)

---

### BodyFooter( *html target* ), VIRTUAL

**BodyFooter** A virtual placeholder to generate HTML BODY footer code.

*html target* The label of the HtmlClass object that writes the HTML code.

The **BodyFooter** method is a virtual placeholder to generate HTML BODY footer code.

BodyFooter is a VIRTUAL method so that other base class methods can directly call the BodyFooter virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreatePageFooter method calls the BodyFooter method.

The WebWindowClass.BodyFooter method does nothing. It is a placeholder for use by derived classes. In particular, the Internet Connect Templates use this method to support HTML code embedded in Web-enabled Clarion applications.

Example:

```
WebWindowClass.CreatePageFooter PROCEDURE(*HtmlClass Target)

CODE

IF SELF.AllowJava
    Target.WriteSubmitApplet(SELF.TimerDelay*1000, SELF.TimerAction)
END
Target.WriteFormFooter
IF SELF.AllowJava
    Target.WriteJavaScript
END
SELF.BodyFooter(Target)
IF (SELF.IsCentered)
    Target.WriteLine('<</CENTER>')
END
Target.WriteLine('<</BODY>')

Target.WriteLine('<</HTML>')
```

See Also: **CreatePageFooter**

## BodyHeader (generate HTML BODY header)

### BodyHeader( *html target* ), VIRTUAL

**BodyHeader** A virtual placeholder to generate HTML BODY header code.

*html target* The label of the HtmlClass object that writes the HTML code.

The **BodyHeader** method is a virtual placeholder to generate HTML BODY header code.

BodyHeader is a VIRTUAL method so that other base class methods can directly call the BodyHeader virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreatePageHeader method calls the BodyHeader method.

The WebWindowClass.BodyHeader method does nothing. It is a placeholder for use by derived classes. In particular, the Internet Connect Templates use this method to support HTML code embedded in Web-enabled Clarion applications.

Example:

```
MyWebWindowClass.CreatePageHeader PROCEDURE(*HtmlClass Target)

CODE
Target.WriteLine('<<HTML>')
Target.WriteFrameCheckScript()
Target.WriteLine('<<HEAD>')
Target.Write('<<TITLE>')
SELF.TitleContents(Target)
Target.WriteLine('<</TITLE>')
Target.WriteLine('<</HEAD>')
Target.Write('<<BODY')
IF (SELF.PageBackground <> COLOR:None)
    Target.Write(' BGCOLOR="" & IC:ColorText(SELF.PageBackground) & ""')
END
Target.WriteLine(' onLoad="setuptimer()" onUnload="killtimer()">')
Target.WriteLine('<<CENTER>')
Target.WriteFormHeader()
SELF.BodyHeader(Target)
```

See Also: [CreatePageHeader](#)

## CreateChildHtml (create HTML for control's children)

**CreateChildHtml**( *html target*, *parent control* [, *border width*] ), **VIRTUAL**

**CreateChildHtml** Generates HTML code for each child control of the specified *parent control*.

*html target* The label of the HtmlClass object that writes the HTML code.

*parent control* An integer constant, variable, EQUATE, or expression containing the parent control's field equate number, or zero (0) if the WINDOW is the parent.

*border width* An integer constant, variable, EQUATE, or expression containing the border width for the parent control. If omitted, border width defaults to zero (0).

The **CreateChildHtml** method generates HTML code for each child control of the parent control. The generated HTML code includes JavaScript where appropriate.

The WebWindowClass uses the CreateChildHtml method to generate HTML code for each parent control within the WINDOW, until the entire WINDOW is represented by the generated HTML.

CreateChildHtml is a VIRTUAL method so that other base class methods can directly call the CreateChildHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreateHtmlPage method calls the CreateChildHtml method.

Example:

```
WebHtmlOptionClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
```

```
SELF.OwnerWindow.CreateChildHtml(Target, SELF.Feq, SELF.GetTableAttributes())
SELF.UpdateCopyChoice
```

## CreateDummyHtmlPage (write empty Html page)

**CreateDummyHtmlPage( *html target*, *filename* )**

### CreateDummyHtmlPage

Generates an empty HTML page.

*html target*      The label of the HtmlClass object that writes the HTML code.

*filename*        A string constant, variable, EQUATE, or expression containing the HTML code filename.

The **CreateDummyHtmlPage** method generates an empty HTML page for transmittal to the client browser.

Implementation:

The WebWindowClass uses the CreateDummyHtmlPage for circumstances where the client browser requests a *partial* page, but the Server determines that a *full* page is required. By sending a dummy page, the Server elicits a full page request from the client browser.

Example:

```
MyWebWindowClass.TakeCreatePage      PROCEDURE

Client                    &WebClientManagerClass
Filename                 CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
  Filename = SELF.Files.GetFilename(Content:Html)
  IF (SELF.Server.GetRequestedWholePage())
    Client.NextHtmlPage
    SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
  ELSE
    Filename = SELF.Files.GetPublicDirectory() & 'dummy.htm'
    SELF.CreateDummyHtmlPage(SELF.HtmlTarget, Filename)
  END
Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
ELSE
  Client.Js1.OpenChannel(SELF.GetTargetSecurity(), SELF.Files)
  SELF.CreateJs1Data(Client.Js1)
  Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()
```

## CreateHtmlPage (generate HTML for a window)

**CreateHtmlPage**( *html target*, *filename* )

<b>CreateHtmlPage</b>	Generates HTML code representing the WINDOW.
<i>html target</i>	The label of the HtmlClass object that writes the HTML code.
<i>filename</i>	A string constant, variable, EQUATE, or expression containing the filename of the file containing the HTML code.

The **CreateHtmlPage** method generates HTML code representing the entire WINDOW. The generated HTML code includes intelligent arrangement of the controls on the resulting Web page, plus JavaScript (Java applet controls) where appropriate.

Implementation: The CreateHtmlPage method generates a page header and footer surrounding the HTML code for all the window controls. The CreateHtmlPage method sets the SentHtml property to one (True).

Example:

```
MyWebWindowClass.TakeCreatePage      PROCEDURE

Client          &WebClientManagerClass
Filename        CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
  Filename = SELF.Files.GetFilename(Content:Html)
  IF (SELF.Server.GetRequestedWholePage())
    Client.NextHtmlPage
    SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
  ELSE
    Filename = SELF.Files.GetPublicDirectory() & 'dummy.htm'
    SELF.CreateDummyHtmlPage(SELF.HtmlTarget, Filename)
  END
Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
ELSE
  Client.Js1.OpenChannel(SELF.GetTargetSecurity(), SELF.Files)
  SELF.CreateJs1Data(Client.Js1)
  Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()
```

See Also: **SentHtml**

## CreateJsldata (generate Java Support Library data)

### CreateJsldata( JSL manager )

**CreateJsldata** Generates Java Support Library (JSL) data for the window.

*JSL manager* The label of the JsldataManagerClass object that generates the JSL data.

The **CreateJsldata** method generates Java Support Library protocol and data for the Java applet controls on the window. This allows very fast partial updates to the Web page.

Implementation: The CreateJsldata method calls the CreateJsldata method for each visible control in the window.

Example:

```
MyWebWindowClass.TakeCreatePage      PROCEDURE

Client                                &WebClientManagerClass
Filename                              CSTRING(FILE:MaxFilePath),AUTO

CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
  Filename = SELF.Files.GetFilename(Content:Html)
  IF (SELF.Server.GetRequestedWholePage())
    Client.NextHtmlPage
    SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
  ELSE
    Filename = SELF.Files.GetPublicDirectory() & 'dummy.htm'
    SELF.CreateDummyHtmlPage(SELF.HtmlTarget, Filename)
  END
Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
ELSE
  Client.Jsl.OpenChannel(SELF.GetTargetSecurity(), SELF.Files)
  SELF.CreateJsldata(Client.Jsl)
  Client.Jsl.CloseChannel
END
SELF.Server.TakePageSent()
```

See Also: [WebControlClass.CreateJsldata](#)

## CreatePageFooter (generate HTML page footer)

---

### CreatePageFooter( *html target* )

#### CreatePageFooter

Generates the HTML page footer for the window.

*html target*

The label of the HtmlClass object that writes the HTML code.

The **CreatePageFooter** method generates the HTML page footer, including any page-ending JavaScript needed by Java controls.

Example:

```
WebWindowClass.CreateHtmlPage PROCEDURE(*HtmlClass Target, STRING HtmlFilename)
CODE
Target.CreateOpen(HtmlFilename,SELF.HtmlOption,SELF.Server.JavaLibraryPath,SELF.Server.Client)
SELF.CreatePageHeader(Target)
SELF.CreateChildHtml(Target, 0, SELF.GetTableAttributes())
SELF.CreatePageFooter(Target)
Target.Close
SELF.SentHtml = TRUE
```

## CreatePageHeader (generate HTML page footer)

---

### CreatePageHeader( *html target* )

**CreatePageHeader** Generates the HTML page header for the window.

*html target*

The label of the HtmlClass object that writes the HTML code.

The **CreatePageHeader** method generates the HTML page header, including any setup JavaScript needed by Java controls.

Example:

```
WebWindowClass.CreateHtmlPage PROCEDURE(*HtmlClass Target, STRING HtmlFilename)
CODE
Target.CreateOpen(HtmlFilename,SELF.HtmlOption,SELF.Server.JavaLibraryPath,SELF.Server.Client)
SELF.CreatePageHeader(Target)
SELF.CreateChildHtml(Target, 0, SELF.GetTableAttributes())
SELF.CreatePageFooter(Target)
Target.Close
SELF.SentHtml = TRUE
```

## CreateUnauthorizedPage (create unauthorized user page)

---

**CreateUnauthorizedPage**( *html target*, *filename* )

### **CreateUnauthorizedPage**

Generates an HTML page showing an access denied message.

*html target*      The label of the HtmlClass object that writes the HTML code.

*filename*        A string constant, variable, EQUATE, or expression containing the filename of the file containing the HTML code.

The **CreateUnauthorizedPage** method generates an HTML page showing an access denied message.

CreateUnauthorizedPage is a VIRTUAL method so that other base class methods can directly call the CreateUnauthorizedPage virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

The default message is “Access denied. Click here to return to previous screen, or wait for 10 seconds.” This default page is installed by default to the \LIBSRC\ICUNAUTH.HTM.

The SetPassword method sets the Authorize property to true, to password protect the Web page (AuthorizeArea). It also sets the authorized password for the AuthorizeArea. The GetAuthorized method collects and validates the end user’s password for the AuthorizeArea. The ValidatePassword method verifies the password entered by the end user. Finally, the TakeCreatePage method generates an “access denied” page (CreateUnauthorizedPage) for invalid passwords.

**Example:**

```
MyWebWindowClass.TakeCreatePage PROCEDURE
Client          &WebClientManagerClass
Filename        CSTRING(FILE:MaxFilePath),AUTO
CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Authorize AND NOT SELF.GetAuthorized())
    Filename = SELF.Files.GetFilename(Content:Unauthorized)
    SELF.CreateUnauthorizedPage(SELF.HtmlTarget, Filename)
    Client.TakeUnauthorized(Filename, SELF.GetTargetSecurity())
    SELF.AuthorizeFailed = TRUE
    Post(EVENT:CloseWindow)
ELSE
    Filename = SELF.Files.GetFilename(Content:Html)
    Client.NextHtmlPage
    SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
    Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
END
SELF.Server.TakePageSent()
```

See Also: **Authorize, AuthorizeArea, GetAuthorized, TakeCreatePage, SetPassword, ValidatePassword**

## GetAuthorized (check user authorization)

---

### GetAuthorized, BYTE

The **GetAuthorized** method collects the username and password and validates it, then returns a value indicating whether the end user is authorized to proceed. A return value of one (1) indicates the end user is authorized; a return value of zero (0) indicates the end user is not authorized.

Implementation:

The **SetPassword** method sets the **Authorize** property to true, to password protect the Web page (**AuthorizeArea**). It also sets the authorized password for the **AuthorizeArea**. The **GetAuthorized** method collects and validates the end user's password for the **AuthorizeArea**. The **ValidatePassword** method verifies the password entered by the end user. Finally, the **TakeCreatePage** method generates an "access denied" page (**CreateUnauthorizedPage**) for invalid passwords.

Return Data Type:

BYTE

Example:

```
MyWebWindowClass.TakeCreatePage PROCEDURE
Client          &WebClientManagerClass
Filename        CSTRING(FILE:MaxFilePath),AUTO
CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Authorize AND NOT SELF.GetAuthorized())
    Filename = SELF.Files.GetFilename(Content:Unauthorized)
    SELF.CreateUnauthorizedPage(SELF.HtmlTarget, Filename)
    Client.TakeUnauthorized(Filename, SELF.GetTargetSecurity())
    SELF.AuthorizeFailed = TRUE
    Post(EVENT:CloseWindow)
ELSE
    Filename = SELF.Files.GetFilename(Content:Html)
    Client.NextHtmlPage
    SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
    Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
END
SELF.Server.TakePageSent()
```

See Also:

**Authorize, AuthorizeArea, CreateUnauthorizedPage, TakeCreatePage, SetPassword, ValidatePassword**

## GetBackgroundColor (return Web window background color)

**GetBackgroundColor**( [*default color*] ), LONG, VIRTUAL

### GetBackgroundColor

Returns the background color of the Web page window.

*default color* An integer variable, constant, EQUATE, or expression containing the color to return if there is no background color. If omitted, *default color* defaults to Color:None.

The **GetBackgroundColor** method returns the background color of the Web page window.

GetBackgroundColor is a VIRTUAL method so that other base class methods can directly call the GetBackgroundColor virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetBackgroundColor method returns the value of the Background property if it contains a color; otherwise it returns the COLOR attribute of the WINDOW.

EQUATEs for the *default color* parameter are declared in \LIBSRC\EQUATES.CLW.

Return Data Type: LONG

Example:

```
WebAreaClass.GetBackgroundColor      PROCEDURE
CODE
IF (SELF.Background <> COLOR:None)
    RETURN SELF.Background
END
RETURN SELF.OwnerWindow.GetBackgroundColor()
```

See Also: **Background**

## GetBackgroundImage (return Web page wallpaper)

---

### GetBackgroundImage, STRING, VIRTUAL

The **GetBackgroundImage** method returns the filename of the file containing the background image of the Web page window.

GetBackgroundImage is a VIRTUAL method so that other base class methods can directly call the GetBackgroundImage virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetBackgroundImage method returns the value of the BackImage property.

Return Data Type: STRING

See Also: BackImage

## GetButtonInClientArea (return button present indicator)

---

### GetButtonInClientArea, SIGNED

The **GetButtonInClientArea** method returns a value indicating whether there are any visible (enabled and not hidden) buttons within the client area. A return value of zero (0) indicates no visible buttons; any other value indicates the presence of a visible button.

Implementation: The **GetButtonInClientArea** returns the control number of the first visible button in the client area. The **GetCreateClose** method calls the **GetButtonInClientArea** method to determine whether to create a close button for the window.

Return Data Type: SIGNED

Example:

```
WebWindowClass.GetCreateClose    PROCEDURE
CODE
CASE SELF.CreateClose
OF CLOSE:Never
    RETURN FALSE
OF CLOSE:IfSystem
    RETURN 0{PROP:System}
OF CLOSE:SystemNoButton
    IF 0{PROP:System} AND NOT SELF.GetButtonInClientArea()
        RETURN TRUE
    END
    RETURN FALSE
OF CLOSE:Always
    RETURN TRUE
END
```

See Also: **GetCreateClose, CreateClose**

## GetChildren (return all child controls)

**GetChildren**( *children*, *parent control* [, *control type*] ), **VIRTUAL**

<b>GetChildren</b>	Returns information about the child controls of the specified parent control.
<i>children</i>	The label of a structure to contain the information about the child controls.
<i>parent control</i>	An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.
<i>control type</i>	An integer constant, variable, EQUATE, or expression indicating the type of child control to include, or exclude. A negative control type excludes controls of the specified type. A positive control type includes controls of the specified type. If omitted, all control types are included.

The **GetChildren** method returns information about the child controls of the specified parent control. The information may be limited to controls of a specific type (for example, TAB controls only), or it may be limited controls excluding a specific type (for example, all controls except TAB controls).

The WebWindowClass object uses the GetChildren method to identify related controls for various purposes, including appropriate positioning of child controls in relation to their parent controls.

GetChildren is a VIRTUAL method so that other base class methods can directly call the GetChildren virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The *children* parameter names a QUEUE with the same structure as the WebControlQueue declared in ICWINDOW.INC as follows:

```
WebControlQueue  QUEUE,TYPE
Feq              SIGNED
ThisControl      &WebControlClass
END
```

EQUATEs for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE.:

Example:

```
WebWindowClass.CreateChildHtml PROCEDURE|
(*HtmlClass Target,SIGNED ParentFeq,STRING Style)
Controls      WebControlQueue
CODE
SELF.GetChildren(Controls, ParentFeq)
SELF.CreateHtml(Controls, Target, Style, SELF.SnapX, SELF.SnapY)
```

## GetControl (return control information)

**GetControl( control [, control information ] ), BYTE, VIRTUAL**

<b>GetControl</b>	Returns information about the specified <i>control</i> .
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control's field equate number.
<i>control information</i>	The label of a structure to contain information about the specified <i>control</i> . If omitted, GetControl returns a value indicating whether the WebWindowClass already "knows" about the control. A return value of one (1) indicates the <i>control</i> is known; a return value of zero (0) indicates the <i>control</i> is unknown.

The **GetControl** method returns information about the specified *control*. If the *control information* parameter is present, GetControl fills it with the control information. If the *control reference* parameter is omitted, GetControl returns a true or false value indicating whether the WebWindowClass object is "aware" of the *control*.

GetControl is a VIRTUAL method so that other base class methods can directly call the GetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The *control information* parameter names a GROUP with the same structure as the WebControlRefGroup GROUP declared in ICWINDOW.INC as follows:

```
WebControlRefGroup  GROUP,TYPE
Control             &WebControlClass
END
```

If the *control information* parameter is present, GetControl calls the GetControlInfo method to set the reference to the WebControlClass object.

Return Data Type: **BYTE**

Example:

```
WebControlClass.GetParentBackgroundColor  PROCEDURE
ParentControlGroup  GROUP(WebControlRefGroup)
END

CODE
SELF.OwnerWindow.GetControl(SELF.RealParentFeq, ParentControlGroup)
IF (NOT ParentControlGroup.Control &= NULL)
RETURN ParentControlGroup.Control.GetBackgroundColor()
END
RETURN SELF.OwnerWindow.GetBackgroundColor()
```

See Also: **GetControlInfo**

## GetControlInfo (return control reference)

---

### GetControlInfo( *control* ), &WebControlClass, VIRTUAL

**GetControlInfo** Returns a reference to the specified *control*.  
*control* An integer constant, variable, EQUATE, or expression containing the control's field equate number.

The **GetControlInfo** method returns a reference to the specified *control*. If the WebWindowClass object is “unaware” of the *control*, GetControlInfo returns a NULL reference.

GetControlInfo is a VIRTUAL method so that other base class methods can directly call the GetControlInfo virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebControlClass maintains a QUEUE of WebControlClass objects and their corresponding control (field equate) numbers. The GetControlInfo method simply returns a reference to the specified WebControlClass object.

Return Data Type: &WebControlClass

Example:

```
WebWindowClass.GetControl PROCEDURE(SIGNED Feq, *WebControlRefGroup Result)
CODE
Result.Control &= SELF.GetControlInfo(Feq)
```

## GetCreateClose (return close button flag)

### GetCreateClose, BYTE, VIRTUAL

The **GetCreateClose** method returns a value indicating whether to create a close button on the Web page. A return value of one (1) indicates a close button should be created; a return value of zero (0) indicates a close button should not be created.

GetCreateClose is a VIRTUAL method so that other base class methods can directly call the GetCreateClose virtual method in a derived class. This lets you easily implement your own custom version of this method.

**Implementation:** The **ResetFromControls** method calls the **GetCreateClose** method to determine whether to create the close button. The **GetCreateClose** method considers the **CreateClose** property and the state of the window (presence of system menu or other enabled buttons) in order to return the appropriate value.

**Return Data Type:** BYTE

**Example:**

```
MyWebWindowClass.ResetFromControls  PROCEDURE
CurFeq          SIGNED,AUTO
CODE
CurFeq = 0{PROP:nextfield}
LOOP WHILE (CurFeq)
  IF (CurFeq)
    SELF.AddControl(CurFeq, IC:GetControlType(CurFeq))
  END
  CurFeq = 0{PROP:nextfield, CurFeq}
END
CurFeq = 04000H
LOOP WHILE (CurFeq{PROP:type})
  SELF.AddControl(CurFeq, IC:GetControlType(CurFeq))
  CurFeq += 1
END
SELF.AddControl(FEQ:ClientArea, CREATE:ClientArea)
IF (SELF.GetCreateClose())
  SELF.AddControl(FEQ:Close, CREATE:Close)
END
SELF.SetChildDefaults
```

**See Also:** CreateClose, ResetFromControls

## GetFirstChild (return first child control)

**GetFirstChild**( *parent control* [, *control type*] ), **SIGNED, VIRTUAL**

<b>GetFirstChild</b>	Returns the control number of the first visible child of the specified parent control and control type.
<i>parent control</i>	An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.
<i>control type</i>	An integer constant, variable, EQUATE, or expression containing the type of child control for which to search. If omitted, any control type is valid.

The **GetFirstChild** method returns the control number of the first “visible” child control of the specified parent control and control type. A visible control is one that is not hidden or disabled.

The WebWindowClass uses the GetFirstChild method primarily to determine whether an area of the Web page has no visible controls and can therefore be omitted. For example, if all toolbar controls are disabled, then the toolbar itself can be omitted from the Web page.

GetFirstChild is a VIRTUAL method so that other base class methods can directly call the GetFirstChild virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The GetShowToolbar and GetShowMenubar methods call the GetFirstChild method. The GetFirstChild method calls the WebControlClass.GetIsChild method for each control.

EQUATEs for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE:.

Return Data Type:

**SIGNED**

Example:

```
WebHtmlMenuClass.GetVisible    PROCEDURE
CODE
IF (PARENT.GetVisible() AND SELF.OwnerWindow.GetFirstChild(SELF.Feq))
    RETURN TRUE
END
RETURN FALSE
```

See Also:

GetShowMenubar, GetShowToolbar, WebControlClass.GetIdChild

## GetHelpHandler (return HTML to show help document)

---

### GetHelpHandler, STRING, VIRTUAL

The **GetHelpHandler** method returns the HTML to request the help document associated with the Web page.

GetHelpHandler is a VIRTUAL method so that other base class methods can directly call the GetHelpHandler virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetHelpHandler method writes HTML to invoke the JavaScript ShowHelp function which submits a request to the Server for a specific help document or document section. The GetHelpReference method supplies the help document to use.

Return Data Type: **STRING**

Example:

```
WebHtmlButtonClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE
Target.Write('<<INPUT TYPE=SUBMIT VALUE="')
Target.Write(IC:QuoteText(SELF.GetText(), IC:RESET:HotValue) & '")
Target.Write(SELF.GetNameAttribute(Target))
IF SELF.Feq{PROP:std} = STD:Help
    Target.Write(SELF.OwnerWindow.GetHelpHandler())
END
Target.WriteLine('>')
```

See Also: **GetHelpReference**

## GetHelpReference (return HTML help document reference)

### GetHelpReference, STRING, VIRTUAL

The **GetHelpReference** method returns the HTML reference for a specific help display. The returned value depends on the values of the HelpDocument and HelpRelative properties, and the WINDOW's HLP attribute.

GetHelpReference is a VIRTUAL method so that other base class methods can directly call the GetHelpReference virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

If HelpRelative is true, GetHelpReference returns a reference like:

```
Document#HelpId
```

otherwise, GetHelpReference returns a reference like:

```
URL/HelpId.htm
```

Where Document or URL comes from the HelpDocument property and HelpId is derived from the HLP attribute for the WINDOW. HelpId is derived by discarding any HLP attribute characters except A-Z, a-z, 0-9, and underscore (\_).

Return Data Type:

STRING

Example:

```
WebWindowClass.GetHelpHandler      PROCEDURE
CODE
RETURN ' onClick="" & IC:QuoteJs1('ShowHelp('' & |
      SELF.GetHelpReference() & '','' & SELF.HelpStyle & ''')') & ''

WebWindowClass.GetHelpTarget      PROCEDURE
CODE
RETURN ' HREF="" & SELF.GetHelpReference() & ''
```

See Also:

WebWindowBaseClass.HelpDocument, WebWindowBaseClass.HelpRelative

## GetHelpTarget (return HTML HREF for help display)

---

### GetHelpTarget, STRING, VIRTUAL

The **GetHelpTarget** method returns the HTML <HREF> for a specific help display.

GetHelpTarget is a VIRTUAL method so that other base class methods can directly call the GetHelpTarget virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetHelpTarget method embeds the reference returned by the GetHelpReference method in an HTML <HREF> statement.

Return Data Type: STRING

Example:

```
WebHtmlItemClass.CreateCellContents PROCEDURE(*HtmlClass Target)

CODE
Target.WriteLine('<<NOBR>')
Target.WriteSpace((SELF.GetLevel()-1)*2)
Target.Write('<<A')
IF SELF.Feq{PROP:std} = STD:Help
  Target.Write(' TARGET="_blank"' & SELF.OwnerWindow.GetHelpTarget())
ELSE
  Target.Write(' HREF="' & Target.GetControlReference(SELF.Feq) & "'')
END
Target.Write('>')
Target.WriteLine(SELF.GetQuotedText() & '<</A>')
Target.WriteLine('<</NOBR><<BR>')
```

See Also: [GetHelpReference](#)

## GetMenubarFeq (return menubar control number)

---

### GetMenubarFeq, SIGNED, VIRTUAL

The **GetMenubarFeq** method returns the control number (field equate) of the window's MENUBAR.

GetMenubarFeq is a VIRTUAL method so that other base class methods can directly call the GetMenubarFeq virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The GetMenubarFeq method returns the value of the MenubarFeq property.

Return Data Type:     **SIGNED**

Example:

```
GetParentFeq     PROCEDURE(SIGNED Feq, *WebWindowBaseClass OwnerWindow)
ParentFeq        SIGNED,AUTO
CODE
ParentFeq = Feq{PROP:Parent}
IF (ParentFeq = 0)
CASE (Feq{PROP:Type})
OF CREATE:MENU
OROF CREATE:ITEM
  ParentFeq = OwnerWindow.GetMenubarFeq()
ELSE
  IF (Feq{PROP:intoolbar})
    ParentFeq = OwnerWindow.GetToolbarFeq()
  ELSE
    ParentFeq = FEQ:ClientArea
  END
END
END
RETURN ParentFeq
```

See Also:             WebWindowBaseClass.MenubarFeq

## GetPageImage (return Web page wallpaper)

---

### GetPageImage, STRING, VIRTUAL

The **GetPageImage** method returns the filename of the file containing the background image of the Web page.

GetPageImage is a VIRTUAL method so that other base class methods can directly call the GetPageImage virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetPageImage method returns the value of the PageImage property.

Return Data Type: STRING

See Also: PageImage

## GetShowMenubar (return menubar include/omit flag)

---

### GetShowMenubar, BYTE, VIRTUAL

The **GetShowMenubar** method returns a value indicating whether to include a menubar area on the Web page.

GetShowMenubar is a VIRTUAL method so that other base class methods can directly call the GetShowMenubar virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetShowMenubar method calls the GetFirstChild method to determine if the Web page menubar is empty.

Return Data Type: BYTE

Example:

```
WebMenubarClass.GetVisible      PROCEDURE  
CODE  
RETURN SELF.OwnerWindow.GetShowMenubar()
```

See Also: GetFirstChild

## GetShowToolbar (return toolbar include/omit flag)

---

### GetShowToolbar, BYTE, VIRTUAL

The **GetShowToolbar** method returns a value indicating whether to include a toolbar on the Web page.

GetShowToolbar is a VIRTUAL method so that other base class methods can directly call the GetShowToolbar virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetShowToolbar method calls the GetFirstChild method to determine if the Web page toolbar is empty.

Return Data Type: **BYTE**

Example:

```
WebToolbarClass.GetVisible      PROCEDURE  
CODE  
RETURN SELF.OwnerWindow.GetShowToolbar()
```

See Also: **GetFirstChild**

## GetTableAttributes (return window HTML <STYLE>)

---

### GetTableAttributes, STRING, VIRTUAL

The **GetTableAttributes** method returns the HTML <STYLE> string for the HTML <TABLE> representing the WINDOW.

GetTableAttributes is a VIRTUAL method so that other base class methods can directly call the GetTableAttributes virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The <STYLE> returned by GetTableAttributes includes the <TABLE> <BORDER> (BorderWidth property), <BGCOLOR> (Background property), and <BACKGROUND> (BackImage property).

Return Data Type: **STRING**

Example:

```
WebWindowClass.CreateHtmlPage      PROCEDURE(*HtmlClass Target, STRING HtmlFilename)
    CODE

    Target.CreateOpen(HtmlFilename,SELF.HtmlOption,SELF.Server.JavaLibraryPath,SELF.Server.Client)
    SELF.CreatePageHeader(Target)
    SELF.CreateChildHtml(Target, 0, SELF.GetTableAttributes())
    SELF.CreatePageFooter(Target)
    Target.Close
    SELF.SentHtml = TRUE
```

See Also: **Background, BackImage, BorderWidth**

## GetTargetSecurity (return public or secure flag)

### GetTargetSecurity, SIGNED

The **GetTargetSecurity** method returns a value indicating whether to use a secure channel or a public channel. A return value of Secure:Full indicates a secure channel; and a return value of Secure:None indicates a public directory.

The WebWindowClass uses this method to develop appropriate pathnames for temporary files and to support secure channels as needed.

Implementation: The GetTargetSecurity method evaluates the IsSecure property and returns Secure:Full if IsSecure is True.

EQUATEs for GetTargetSecurity return values are declared in ICFILES.INC as follows:

```

                                ITEMIZE,PRE(Secure)
Default                          EQUATE
None                             EQUATE
Full                             EQUATE
Last                             EQUATE(Secure:Full)
                                END

```

Return Data Type: **SIGNED**

Example:

```

MyWebWindowClass.TakeCreatePage PROCEDURE
Client                          &WebClientManagerClass
Filename                        CSTRING(FILE:MaxFilePath),AUTO
CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Authorize AND NOT SELF.GetAuthorized())
    Filename = SELF.Files.GetFilename(Content:Unauthorized)
    SELF.CreateUnauthorizedPage(SELF.HtmlTarget, Filename)
    Client.TakeUnauthorized(Filename, SELF.GetTargetSecurity())
    SELF.AuthorizeFailed = TRUE
    Post(EVENT:CloseWindow)
ELSE
    Filename = SELF.Files.GetFilename(Content:Html)
    Client.NextHtmlPage
    SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
    Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
END
SELF.Server.TakePageSent()

```

See Also: **IsSecure**

## GetToolBarFeq (return Toolbar control number)

---

### GetToolBarFeq, SIGNED, VIRTUAL

The **GetToolBarFeq** method returns the control number (field equate) of the window's TOOLBAR.

GetToolBarFeq is a VIRTUAL method so that other base class methods can directly call the GetToolBarFeq virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetToolBarFeq method returns the value of the ToolbarFeq property.

Return Data Type: SIGNED

Example:

```

GetParentFeq    PROCEDURE(SIGNED Feq, *WebWindowBaseClass OwnerWindow)
ParentFeq      SIGNED,AUTO
CODE
ParentFeq = Feq{PROP:Parent}
IF (ParentFeq = 0)
CASE (Feq{PROP:Type})
OF CREATE:MENU
OROF CREATE:ITEM
ParentFeq = OwnerWindow.GetMenubarFeq()
ELSE
IF (Feq{PROP:intoolbar})
ParentFeq = OwnerWindow.GetToolBarFeq()
ELSE
ParentFeq = FEQ:ClientArea
END
END
END
RETURN ParentFeq

```

See Also: WebWindowBaseClass.ToolbarFeq

## GetToolBarMode (return toolbar entity)

---

### GetToolBarMode, SIGNED, VIRTUAL

The **GetToolBarMode** method returns a value indicating what the toolbar is driving. For example, a **BrowseBox**, a **Relation Tree**, or an update form. The **WebWindowClass** uses this method to decide whether a toolbar button warrants a full page update or a partial page update.

**GetToolBarMode** is a **VIRTUAL** method so that other base class methods can directly call the **GetToolBarMode** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

**EQUATEs** for **GetToolBarMode** return values are declared in **TPLEQU.CLW** as follows:

```
FormMode      EQUATE(1)
BrowseMode    EQUATE(2)
TreeMode      EQUATE(3)
```

Return Data Type:

**BYTE**

Example:

```
WebJavaToolBarClass.GetEventAction    PROCEDURE(SIGNED EventNo)

CODE
IF (EventNo = EVENT:Accepted)
  IF (SELF.Ownerwindow.GetToolBarMode() = FormMode)
    RETURN Update:Full
  ELSE
    RETURN Update:Partial
  END
END
RETURN Update:OnBrowser
```

## GetWebActiveFrame (return WebFrameClass reference)

---

### GetWebActiveFrame, WebFrameClass, VIRTUAL

The **GetWebActiveFrame** method returns a reference to the active WebFrameClass. The WebWindowClass uses this method to pass events to the WebFrameClass for processing.

GetWebActiveFrame is a VIRTUAL method so that other base class methods can directly call the GetWebActiveFrame virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type:      WebFrameClass

Example:

```
WebMenuBaseClass.GetPosition PROCEDURE(*SIGNED x,*SIGNED y,*SIGNED width,*SIGNED height)
```

```
ActiveFrame    &WebFrameClass
```

```
CODE
ActiveFrame &= SELF.OwnerWindow.GetWebActiveFrame()
IF SELF.ParentFeq=ActiveFrame.GetMenubarFeq()
  x = SELF.Feq{PROP:childindex} * 40
  y = 0
ELSE
  x = 0
  y = SELF.Feq{PROP:childindex} * 40
END
width = 10
height = 10
```

See Also:              WebFrameClass

## Init (initialize the WebWindowClass object)

---

**Init**( *server*, *html target* [,*area*] )

<b>Init</b>	Initializes the WebWindowClass object.
<i>server</i>	The label of the WebServerClass object that establishes communications with the Application Broker and handles the information coming through the channel.
<i>html target</i>	The label of the HtmlClass object that writes the HTML code representing the window.
<i>area</i>	A string constant, variable, EQUATE, or expression containing the identity of a Web page generated by the Server application that may be password protected. The Client browser displays this value when prompting the end user for an authorized username and password.

The **Init** method initializes the WebWindowClass object.

**Implementation:** The Init method sets the default values of all the WebWindowClass properties, including the properties that control HTML code generation. You may override these default values by subsequent (after init method) assignments to the WebWindowClass properties.

**Example:**

```
PrepareProcedure ROUTINE
  WebWindow.Init(WebServer, HtmlManager, window{PROP:text} & ' (BrowseCustomer)')
!etc.
```

**See Also:** AllowJava, Authorize, AuthorizeArea, Background, BackImage, BorderWidth, CloseImage, CreateCaption, CreateClose, CreateToolbar, DisabledAction, Files, FormatBorderWidth, GroupBorderWidth, HtmlTarget, IsCentered, MenubarType, OptionBorderWidth, PageBackground, PageImage, Server, SheetBorderWidth, TimerDelay, TimerAction, IsSecure

## Kill (shut down the WebWindowClass object)

---

### Kill, VIRTUAL

The **Kill** method frees any memory allocated during the life of the object and performs any other necessary termination code.

Kill is a VIRTUAL method so that other base class methods can directly call the Kill virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
ProcedureReturn ROUTINE
  WebWindow.Kill
!etc.
```

## ResetFromControls (set control information)

---

### ResetFromControls

The **ResetFromControls** method dynamically builds the WebWindowClass object's knowledge of the WINDOW's controls and pseudo-controls from the present state of the WINDOW.

You may call this method to accomodate dynamically created or destroyed controls.

Implementation:

The ResetFromControls method calls the AddControl method for each control and pseudo-control in the opened WINDOW. When all controls are set, ResetFromControls establishes relationships between controls by calling the SetParentDefaults method.

Example:

```
WebWindowClass.TakeEvent PROCEDURE
  CODE
  !procedure code
  CASE (EVENT())
  OF EVENT:NewPage
    IF (SELF.Server.GetReadyForPage())
      SELF.TakeCreatePage
    END
  OF EVENT:OpenWindow
    SELF.ResetFromControls
  END
!etc.,
```

See Also: [AddControl](#), [SetParentDefaults](#)

## SetBackground (set Web page window background)

---

**SetBackground( [color] [,image] )**

**SetBackground** Sets the Web page window background color and image.

*color* An integer constant, variable, EQUATE, or expression indicating the PAGE color. This color is inherited by the window components, such as the toolbar and client area, unless a specific color is set for the component.

*image* A string constant, variable, EQUATE, or expression containing a filename. The WebWindowClass object displays (tiles) the image in the specified file as the background to the Web page.

The **SetBackground** method sets the Web page window background color and image. Other window components (caption, menubar, toolbar, and client area) inherit the *color* and *image* unless overridden.

Implementation: The SetBackground method sets the Background property and the BackImage property. The GetBackgroundColor method returns the value of the Background property.

Example:

```
PrepareProcedure ROUTINE
  WebWindow.SetBackground(4227200, 'Widgets.IC0')
!etc.
```

See Also: Background, BackImage, GetBackgroundColor

## SetCentered (center window in Web page)

---

### SetCentered( [*center*] )

**SetCentered** Centers or left-justifies the window in the generated Web page.

*center* An integer constant, variable, EQUATE, or expression indicating whether to horizontally center or left-justify the window within the generated Web page. A value of one (1 or True) centers the window; a value of zero (0 or False) left-justifies the window. If omitted, *center* defaults to True.

The **SetCentered** method prepares the WebWindowClass object to center or left-justify the window within the generated Web page, depending on the value of the *center* parameter.

Implementation: The SetCentered method sets the IsCentered property.

Example:

```
PrepareProcedure ROUTINE
  WebWindow.SetCentered(True)
!etc.
```

See Also: IsCentered

## SetChildDefaults (set children of each control)

---

### SetChildDefaults

The **SetChildDefaults** method sets the child controls for each control on the Web page. This positional determination of parent/child relationships results in proper alignment and processing of controls on the Web page.

Implementation: The SetChildDefaults method calls the WebControlClass.SetChildren method for each control on the WINDOW.

Example:

```
WebWindowClass.ResetFromControls PROCEDURE
CurFeg          SIGNED,AUTO
CODE
!procedure code
SELF.SetChildDefaults
```

See Also: WebControlClass.SetChildDefaults

## SetFormatOptions (set Web page scale and alignment)

---

**SetFormatOptions**( *snaptoX*, *snaptoY*, *scaleX*, *scaleY* )

### SetFormatOptions

Sets the control scaling and alignment factors for the Web page.

*snaptoX* An integer constant, variable, EQUATE, or expression containing a horizontal “snap to” factor for aligning controls on the Web page.

*snaptoY* An integer constant, variable, EQUATE, or expression containing a vertical “snap to” factor for aligning controls on the Web page.

*scaleX* An integer constant, variable, EQUATE, or expression containing a horizontal scaling factor for sizing Java controls on the Web page.

*scaleY* An integer constant, variable, EQUATE, or expression containing a vertical scaling factor for sizing Java controls on the Web page.

The **SetFormatOptions** method sets the control scaling and alignment factors for the Web page. The precise effect of these properties is determined by the `LayoutHtmlClass` and `HtmlClass` objects that actually apply the factors.

Implementation: The `SetFormatOptions` method sets the initial value of the `SnapX` and `SnapY` properties, and the initial values of the `HtmlOption` property.

Example:

```
WebWindowClass.Init PROCEDURE(*WebServerClass Server, |
    *HtmlClass HtmlTarget, <STRING AuthorizeArea>)
CODE
!procedure code
SELF.SetFormatOptions(2, 2, 6, 13)
```

See Also: `SnapX`, `SnapY`, `HtmlOption`

## SetHelpDocument (enable single document Web page help)

---

**SetHelpDocument**( *filename*, *style* )

**SetHelpDocument** Lets the generated Web page request topics within a single HTML help document, and lets the Server procedure respond to the help requests.

*filename* A string constant, variable, EQUATE, or expression containing the pathname of an HTML document relative to the Application Broker root directory.

*style* A string constant, variable, EQUATE, or expression containing ...

The **SetHelpDocument** method prepares the WebWindowClass object to generate a Web page that can request page-specific help topics within a single HTML help document, and to respond to the requests by supplying the requested material to the Client.

Implementation:

The SetHelpDocument method sets the HelpDocument property to *filename*, the HelpEnabled property to True, the HelpRelative property to True, and the HelpStyle property to *style*.

The GetHelpReference method derives the help topic from the WINDOW's HLP attribute.

Example:

```
PrepareProcedure ROUTINE
  WebWindow.SetHelpDocument('Widgets/OrdHelp.htm', '')
```

See Also:

GetHelpReference, HelpDocument, HelpEnabled, HelpRelative, HelpStyle

## SetHelpURL (enable multiple document Web page help)

---

### SetHelpURL( *URL*, *style* )

<b>SetHelpURL</b>	Lets the generated Web page request a single HTML help document, and lets the Server procedure respond to the help requests.
<i>URL</i>	A string constant, variable, EQUATE, or expression containing the URL of an HTML document.
<i>style</i>	A string constant, variable, EQUATE, or expression containing ...

The **SetHelpURL** method prepares the WebWindowClass object to generate a Web page that can request a page-specific HTML help document, and to respond to the requests by supplying the requested document to the Client.

Implementation:

The SetHelpURL method sets the HelpDocument property to *URL*, the HelpEnabled property to True, the HelpRelative property to False, and the HelpStyle property to *style*.

The GetHelpReference method derives the help document name from the WINDOW's HLP attribute.

Example:

```
PrepareProcedure ROUTINE  
  WebWindow.SetHelpURL('http://www.MyISP/Widgets', '')
```

See Also:

GetHelpReference, HelpDocument, HelpEnabled, HelpRelative, HelpStyle

## SetPageBackground (set Web page background)

---

**SetPageBackground( [*color*] [,*image*] )**

### **SetPageBackground**

Sets the Web page background color and image.

*color* An integer constant, variable, EQUATE, or expression indicating the PAGE color.

*image* A string constant, variable, EQUATE, or expression containing a filename. The WebWindowClass object displays (tiles) the image in the specified file as the background to the Web page.

The **SetPageBackground** method sets the Web page background color and image.

Implementation: The SetPageBackground method sets the PageBackground property and the PageImage property.

Example:

```
PrepareProcedure ROUTINE
  WebWindow.SetPageBackground(4227200, 'Widgets.ICO')
!etc.
```

See Also: PageBackground, PageImage

## SetPassword (require Web page password)

---

**SetPassword**( *password* [, *case sensitive*] )

<b>SetPassword</b>	Sets the password required to access the Web page representing the WINDOW.
<i>password</i>	A string constant, variable, EQUATE, or expression containing the password.
<i>case sensitive</i>	A Boolean constant, variable, EQUATE, or expression indicating whether password verification is case sensitive.

The **SetPassword** method sets the password required to access the Web page representing the WINDOW.

Implementation:

The SetPassword method sets the Authorize property to true, to password protect the Web page (AuthorizeArea). It also sets the authorized password for the AuthorizeArea. The GetAuthorized method collects and validates the end user's password for the AuthorizeArea. The ValidatePassword method verifies the password entered by the end user. Finally, the TakeCreatePage method generates an "access denied" page (CreateUnauthorizedPage) for invalid passwords.

Example:

```
PrepareProcedure ROUTINE
!routine code
WebWindow.Init(WebServer, HtmlManager, window{PROP:text} & ' (BrowseCustomer)')
WebWindow.SetPassword(UserPassword, FALSE)
```

See Also:

Authorize, AuthorizeArea, CreateUnauthorizedPage, GetAuthorized, TakeCreatePage, ValidatePassword

## SetSplash (make this a splash window)

---

### SetSplash( *timer delay* )

**SetSplash** Marks the window as a splash window.  
*timer delay* A string constant, variable, EQUATE, or expression containing the password.

The **SetSplash** method designates the window is a splash window. By convention, a splash window opens automatically at program startup, providing the end-user with a sense of familiarity and confidence, and perhaps an introduction to the program.

**Tip:** Splash screens shouldn't contain buttons (or regions) that cause them to close. If it does, and the control number (feq) of the button matches a control number on the frame procedure, then the frame control's action is invoked rather than the splash screen control's action.

Implementation: The SetSplash method sets the IsSplash property to true, the TimerAction property to Update:Refresh, and the TimerDelay property to *timer delay*.

Example:

```
Splash PROCEDURE
!procedure data
  CODE
  !procedure code

PrepareProcedure ROUTINE
  !routine code
  WebWindow.SetSplash(10)
```

See Also: IsSplash, TimerAction, TimerDelay

## SetTimer (set Web page timer and action)

**SetTimer**( *timer delay*, *timer action* )

<b>SetTimer</b>	Sets a time interval and the action to take when the time interval expires.
<i>timer delay</i>	An integer constant, variable, EQUATE, or expression indicating the time period for Java applets to wait before initiating the <i>timer action</i> .
<i>timer action</i>	An integer constant, variable, EQUATE, or expression indicating the action the browser takes after the <i>timer delay</i> interval. The action is repeated each time the <i>timer delay</i> interval expires.

The **SetTimer** method sets a time interval and the action to take when the time interval expires. Both the *timer delay* interval and the *timer action* are implemented by the *Client browser*, not the Server.

**Tip:** Use this method sparingly to minimize network traffic.

There are four valid *timer actions*, three of which consist of a request to the Server for an update of the Web page. Update:Full submits any entered items and requests a complete screen redraw. Update:Partial submits any entered items and requests only the data to fill Java controls (see *JSL Manager Class*). Update:Refresh submits no items, but requests a complete screen redraw. Update:OnBrowser updates the Web page as far as possible without contacting the Server.

Implementation: The SetTimer method sets the TimerAction and TimerDelay properties.

EQUATEs for the TimerAction property are declared in ICSTD.INC as follows:

```
Update:OnBrowser    EQUATE(0)
Update:Partial     EQUATE(1)
Update:Full        EQUATE(2)
Update:Refresh     EQUATE(3)    !Splash screen only
```

The Init method sets the initial value of the TimerAction property to zero (0). The SetTimer method sets subsequent values of the TimerAction property.

Example:

```
PrepareProcedure ROUTINE
!routine code
WebWindow.SetTimer(10, Update:Partial)
```

See Also: TimerAction, TimerDelay

## SuppressControl (omit control from Web page)

---

### SuppressControl(*control*)

**SuppressControl** Omits the specified *control* from the Web page.

*control* An integer constant, variable, EQUATE, or expression containing the control's field equate number.

The **SuppressControl** method omits the specified *control* from the generated Web page.

Implementation: The **SuppressControl** method instantiates a **WebNullControlClass** object to represent the *control*. You must call the **SuppressControl** method before the corresponding **AddControl** method; that is, before calling **ResetFromControls**.

Example:

```
PrepareProcedure ROUTINE
!routine code
WebWindow.SuppressControl(?Insert)
WebWindow.SuppressControl(?Change)
WebWindow.SuppressControl(?Delete)
```

See Also: **AddControl**, **ResetFromControls**

## TakeCreatePage (fill Client request for page)

### TakeCreatePage, VIRTUAL

The **TakeCreatePage** method prepares a page in response to a Client request, then notifies the `WebClientManagerClass` object and the `WebServerClass` object that the page is ready.

`TakeCreatePage` is a VIRTUAL method so that other base class methods can directly call the `TakeCreatePage` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `TakeEvent` method calls the `TakeCreatePage` method.

The `TakeCreatePage` method may generate a complete Web page (HTML representation of the WINDOW), a page of JSL data to partially update the Web page, an “access denied” page, or a dummy page if the Client requested a partial page and the Server determines a full page is necessary.

Example:

```

WebWindowClass.TakeEvent          PROCEDURE
CODE
IF (SELF.Server.Active)
CASE SELF.Server.TakeEvent()
OF NET:Request
SELF.TakeRequest
OF NET:Terminate
RETURN 1
END
CASE (EVENT())
OF EVENT:NewPage
IF (SELF.Server.GetReadyForPage())
SELF.TakeCreatePage
END
OF EVENT:OpenWindow
SELF.ResetFromControls
END
WebActiveFrame.TakeEvent
IF (SELF.Server.GetReadyForPage())
POST(EVENT:NewPage)
END
END
RETURN 0

```

See Also: `TakeEvent`

## TakeEvent (handle browser and ACCEPT loop events)

---

### TakeEvent, VIRTUAL

The **TakeEvent** method handles all ACCEPT loop events for the WebWindowClass object, including events channeled from the Client to the Server through the Broker.

A return value of one (1) indicates the calling entity should BREAK out of the current ACCEPT loop. A return value of zero (0) indicates the calling entity should continue through the current ACCEPT loop.

TakeEvent is a VIRTUAL method so that other base class methods can directly call the TakeEvent virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

When the WINDOW is opened, the TakeEvent method builds the WebWindowClass object's knowledge of the WINDOW's controls and pseudo-controls from the present state of the WINDOW.

Thereafter, it processes requests from the Client browser by translating them to appropriate Clarion EVENTS. After the Clarion EVENTS are processed by the ACCEPT loop code, it sends information (HTML code or JSL data) to the Broker for forwarding to the Client.

#### Example:

```
AlmostAny PROCEDURE
!procedure data
CODE
DO PrepareProcedure
ACCEPT
  IF WebWindow.TakeEvent()
    BREAK
  END
CASE EVENT()
!etc
END
DO ProcedureReturn
```

## TakeRequest (process browser event/request)

---

### TakeRequest, VIRTUAL

The **TakeRequest** method handles ACCEPT loop events channeled from the Client browser to the Server through the Broker.

TakeRequest is a VIRTUAL method so that other base class methods can directly call the TakeRequest virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: TakeRequest processes requests from the Client browser by accepting field values from the browser and updating the corresponding Clarion controls' USE variables, then POSTing the associated event to the appropriate control (WebControlClass.ResetControl).

Example:

```
WebWindowClass.TakeEvent  PROCEDURE
CODE
IF (SELF.Server.Active)
CASE SELF.Server.TakeEvent()
OF NET:Request
SELF.TakeRequest
OF NET:Terminate
RETURN 1
END
CASE (EVENT())
OF EVENT:NewPage
IF (SELF.Server.GetReadyForPage())
SELF.TakeCreatePage
END
OF EVENT:OpenWindow
SELF.ResetFromControls
END
WebActiveFrame.TakeEvent
IF (SELF.Server.GetReadyForPage())
POST(EVENT:NewPage)
END
END
RETURN 0
```

See Also: TakeEvent, WebControlClass.ResetControl

## TakeUnknownSubmit (a virtual to handle unexpected requests)

**TakeUnknownSubmit( *control name*, *new value* ),VIRTUAL**

### TakeUnknownSubmit

A virtual placeholder method to handle unexpected Client requests.

*control name* A string constant, variable, EQUATE, or expression containing the HTML control identifier.

*new value* A string constant, variable, EQUATE, or expression containing the new value of the HTML control.

The **TakeUnknownSubmit** method is a virtual placeholder method to handle Client data assignments to unrecognized Web page fields.

TakeUnknownSubmit is a VIRTUAL method so that other base class methods can directly call the TakeUnknownSubmit virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The TakeRequest method calls the TakeUnknownSubmit method.

Example:

```
MyWebWindowClass.TakeRequest PROCEDURE
NextSubmit      &SubmitItemClass
CurFeq         SIGNED,AUTO
Index           SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()
  END
END
LOOP
NextSubmit &= SELF.Server.SetNextAction()
IF (NextSubmit &= NULL) THEN BREAK.
CurFeq = NextSubmit.Feq
CASE (CurFeq)
OF FEQ:UNKNOWN
  SELF.TakeUnknownSubmit(NextSubmit.Name, NextSubmit.NewValue)
ELSE
  IF (SELF.GetControl(CurFeq))
    IF (NextSubmit.Event)
      IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
        SELF.Controls.ThisControl.ResetControl(NextSubmit)
      END
    END
  END
END
END
END
```

See Also: [TakeRequest](#)

## TitleContents (set browser titlebar)

---

### TitleContents( *html object* ),VIRTUAL

**TitleContents** Generates the HTML representation of the WINDOW's titlebar text.

*html target* The label of the HtmlClass object that writes the HTML code representing the window.

The **TitleContents** method generates the HTML representation of the WINDOW's titlebar text, primarily for use in the Client browser's titlebar.

TitleContents is a VIRTUAL method so that other base class methods can directly call the TitleContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The TitleContents method surrounds the WINDOW's PROP:Text value with quotations suitable for the Client browser.

The CreatePageHeader method calls the TitleContents method.

Example:

```
WebWindowClass.CreatePageHeader PROCEDURE(*HtmlClass Target)
```

```
CODE
Target.WriteLine('<<HTML>')
Target.WriteFrameCheckScript()
IF (SELF.TimerAction = Update:Refresh)
    Target.WriteRefreshTimer(SELF.TimerDelay)
END
Target.WriteLine('<<HEAD>')
Target.Write('<<TITLE>')
SELF.TitleContents(Target)
Target.WriteLine('<</TITLE>')
!etc.
```

See Also: CreatePageHeader

## ValidatePassword (verify password)

**ValidatePassword**( *username, password* ), **BYTE. VIRTUAL**

**ValidatePassword** Sets the password required to access the Web page representing the WINDOW.

*username* A string constant, variable, EQUATE, or expression containing the username.

*password* A string constant, variable, EQUATE, or expression containing the password.

The **ValidatePassword** method verifies that the password entered by the end user matches the authorized password for the Web page. **ValidatePassword** ignores the username; however, the Internet Connect Templates provide the *Internet, password validation - data section* and the *Internet, password validation - code section* embed points so you can validate the username if you choose.

**ValidatePassword** is a VIRTUAL method so that other base class methods can directly call the **ValidatePassword** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The **SetPassword** method sets the **Authorize** property to true, to password protect the Web page (**AuthorizeArea**). It also sets the authorized password for the **AuthorizeArea**. The **GetAuthorized** method collects and validates the end user's password for the **AuthorizeArea**. The **ValidatePassword** method verifies the password entered by the end user. Finally, the **TakeCreatePage** method generates an "access denied" page (**CreateUnauthorizedPage**) for invalid passwords.

Return Data Type: **BYTE**

Example:

```
WebWindowClass.GetAuthorized PROCEDURE
Password          STRING(255)
UserName          STRING(255)
CODE
SELF.Server.Broker.GetAuthorizedInfo(SELF.AuthorizeArea, UserName, Password)
RETURN SELF.ValidatePassword(UserName, Password)
```

See Also: **Authorize, AuthorizeArea, CreateUnauthorizedPage, GetAuthorized, TakeCreatePage, SetPassword**

# WEB CONTROL CLASS

<b>Overview</b>	<b>219</b>
WebControlClass Concepts .....	219
Relationship to Other Internet Builder Classes .....	219
Internet Connect Template Implementation .....	220
Source Files .....	220
<b>WebControlClass Properties</b>	<b>221</b>
ActionOnAccept (browser action for control) .....	221
Container (container control) .....	222
DisabledAction (HTML for disabled control) .....	222
Feq (control number) .....	223
IsDynamic (memory allocated flag) .....	223
OwnerWindow (owner window) .....	223
ParentFeq (Web page parent control ) .....	224
RealParentFeq (window parent control) .....	224
<b>WebControlClass Methods</b>	<b>225</b>
Functional Organization—Expected Use .....	225
BeforeResetControl (a virtual for control preprocessing) .....	227
CreateCellContents (write HTML for control) .....	227
CreateCellFooter (end HTML for control attributes) .....	228
CreateCellHeader (begin HTML for control attributes) .....	228
CreateColorParameters (write Java applet color parameters) .....	229
CreateForeColorParameter (write Java applet foreground color parameter) .....	230
CreateHtml (write HTML for control and its attributes) .....	231
CreateHtmlExtra (write HTML for related control) .....	232
CreateJslData (update Web page controls) .....	232
CreateParams (write all parameters for Java control) .....	233
DoSetChildDefaults (set nested child controls) .....	233
GetAlignText (return text alignment information) .....	234
GetAppletType (return applet type) .....	234
GetBackgroundColor (return background color) .....	235
GetCanDisable (return disable-ability flag) .....	236
GetCellAttributes (return control attributes) .....	237
GetChoiceChanged (return selection change) .....	238
GetEventAction (return browser action) .....	239

GetFont (add font information) .....	240
GetHasHotkey (control text contains '&') .....	240
GetIsChild (return family identity) .....	241
GetLevel (return nesting level) .....	242
GetNameAttribute (return HTML control name) .....	242
GetParentBackgroundColor (return parent background color) .....	243
GetPosition (get control coordinates) .....	244
GetQuotedText (return control text in quotes) .....	245
GetTableAttributes (return HTML STYLE).....	245
GetText (return control text) .....	245
GetUseChanged (return contents changed indicator) .....	246
GetVisible (return control status flag) .....	247
Init (initialize the WebControlClass object) .....	248
Kill (shut down the WebControlClass object) .....	249
PopFont (restore pre-PushFont font) .....	250
PushFont (implement control font) .....	251
RefreshDisabled (update Web page control status) .....	252
ResetControl (update server control).....	253
ResetFromQueue (record changes to Server LIST queue).....	254
SetAutoSpotLink (a virtual to set AutoSpotLink) .....	255
SetBorderWidth (a virtual to set BorderWidth) .....	255
SetBreakable (allow word wrap) .....	256
SetChildDefaults (a virtual to set Web page control children) .....	256
SetDescription (a virtual to set AltText) .....	257
SetEventAction (associate browser action with control event) .....	258
SetParentDefaults (confirm parent) .....	260
SetQueue (a virtual to set the FromQ property) .....	261
UpdateCopyChoice (save selected item number) .....	261
UpdateCopyUse (save copy of control contents) .....	262

## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

## WebControlClass Concepts

---

The WebControlClass analyzes a single Clarion control and generates the HTML code to duplicate the Clarion control on a Web page.

The WebControlClass manages the HTML equivalent of a Clarion control, or other window component such as a window titlebar or client area. This class manages control attributes (color, font, position, enable/disable, etc.), parent/child relationships (for positioning and property inheritance purposes), and any special browser events or actions associated with the control. For example, you may associate a “partial page update” action with a check box so that a filter invoked by the check box immediately displays in the Client browser.

The WebControlClass manages the control attributes and behaviors that are common to all Clarion controls. For example, MSG attributes are handled the same for all controls; however, the HTML code generated for a CHECK is different than for an ENTRY. Therefore, each specific type of Clarion control also has its own class derived from the WebControlClass to implement type-specific behavior. For example, the WebHtmlPromptClass represents a PROMPT, and the WebHtmlEntryClass represents an ENTRY, etc. These derived classes are documented in the *WebControlClass Derived Classes* chapter.

## Relationship to Other Internet Builder Classes

---

### HtmlItemClass

The WebControlClass is derived from the HtmlItemClass. The HtmlItemClass is an abstract class that does no real work; it simply provides a means for other classes to reference any of its derived classes. The WebControlClass and its derived control-specific classes (WebHTMLPromptClass, WebHTMLEntryClass, etc.) provide all of the WebControlClass object’s functionality.

### WebWindowClass

The WebWindowClass creates and manages instances of the WebControlClass as needed to generate HTML for each control in the WINDOW.

## LayoutHtmlClass

The `LayoutHtmlClass` uses instances of the `WebControlClass` to optimally format, position, and align the Web page controls on the Web page generated by the `WebWindowClass` object.

## Internet Connect Template Implementation

---

The `WebWindowClass` creates and manages instances of the `WebControlClass` as needed to generate HTML for each control in the WINDOW. Therefore, the template generated code generally only references a few `WebControlClass` objects during procedure initialization, for example:

```
PrepareProcedure ROUTINE
!routine code
IC:CurControl &= WebWindow.AddControl(?Browse:1)
IC:CurControl.SetQueue(Queue:Browse:1)
```

## Source Files

---

The `WebControlClass` source code is installed by default to the `\LIBSRC` folder. The `WebControlClass` declarations are in the following `.INC` files. The corresponding method definitions are in the corresponding `.CLW` file.

ICLAYOUT.INC    `HtmlItemClass`

ICWINDOW.INC   `WebControlClass`

## *WebControlClass Properties*

### **ActionOnAccept (browser action for control)**

---

**ActionOnAccept****BYTE**

The **ActionOnAccept** property indicates the action the browser takes when the Web page control is accepted.

There are three valid actions. Update:Full submits any entered items and requests a complete screen redraw. Update:Partial submits any entered items and requests only the data to fill Java controls (see *JSL Manager Class*). Update:OnBrowser updates the Web page as far as possible without contacting the Server.

**Implementation:**

The Init method sets the initial value of the ActionOnAccept property. The SetEventAction method sets subsequent values of ActionOnAccept. The GetEventAction returns the action the browser takes when the control is accepted.

EQUATEs for the ActionOnAccept property are declared in ICSTD.INC as follows:

```
Update:OnBrowser    EQUATE(0)
Update:Partial      EQUATE(1)
Update:Full         EQUATE(2)
```

Update:Refresh is only valid for windows with a TimerDelay. It is not appropriate for the WebControlClass.ActionOnAccept property.

**See Also:**

Init, GetEventAction, SetEventAction

## Container (container control)

---

### Container      &WebControlClass

The **Container** property is a reference to another WebControlClass object that is the container of this WebControlClass object.

The WebControlClass object uses this property to allow controls to inherit properties from their container controls.

Implementation:      The Init method sets the Container property to NULL.

The IBC Library objects use container controls to group multiple Java controls into a single Java container control. Grouping these controls optimizes the performance of the Server (Web-enabled application). The Server generates and transmits less HTML code, and the containerized Java applets initialize and display faster on the Client machine.

See Also:              Init

## DisabledAction (HTML for disabled control)

---

### DisabledAction    BYTE

The **DisabledAction** property indicates how the disabled control is represented on the Web page. This property lets you determine how to handle controls that are disabled under Windows, but cannot be disabled under a browser, because the HTML control does not support disabling.

Valid actions include, always display the disabled control, always hide the disabled control, and disable the control if HTML supports it, otherwise hide the control.

Implementation:      The Init method sets the initial value of the DisabledAction property to equal the WebWindowClass.DisabledAction property. You may override the DisabledAction for a control with a simple assignment statement.

EQUATEs for the DisabledAction property are declared in ICWINDOW.INC as follows:

```

ITEMIZE,PRE(DISABLE)
Hide            EQUATE            !always hide disabled control
OptHide        EQUATE            !hide if HTML control won't disable
Show           EQUATE            !always show disabled control
END
```

See Also:              Init, WebWindowClass.DisabledAction

## Feq (control number)

---

**Feq**      **SIGNED**

The **Feq** property contains the control's control number (field equate). The `WebWindowClass` uses this property to uniquely identify the control.

Implementation:      The `Init` method sets the value of the `Feq` property.

See Also:              `Init`

## IsDynamic (memory allocated flag)

---

**IsDynamic**      **BYTE**

The **IsDynamic** property indicates whether memory was allocated for the `WebControlClass` object. The `WebWindowClass` object uses this property to free the allocated memory.

Implementation:      The `WebWindowClass.Kill` method `DISPOSES` the `WebControlClass` object based on the value of the `IsDynamic` property.

See Also:              `WebWindowClass.Kill`

## OwnerWindow (owner window)

---

**OwnerWindow**      **&WebWindowBaseClass**

The **OwnerWindow** property is a reference to the `WebWindowClass` object that instantiated this `WebControlClass` object. The `WebControlClass` object uses this property to reference `WebWindowClass` methods and properties.

Implementation:      The `Init` method sets the value of the `OwnerWindow` property.

See Also:              `Init`

## ParentFeq (Web page parent control)

---

### ParentFeq      SIGNED

The **ParentFeq** property contains the control number (field equate) of the parent control for Web page purposes. The `WebWindowClass` uses this property to maintain a position-based parent/child relationship that allows proper positioning and processing of controls on the Web page.

Implementation:      The `SetChildDefaults` method sets the value of the `ParentFeq` property.

See Also:              `SetChildDefaults`

## RealParentFeq (window parent control)

---

### RealParentFeq      SIGNED, PROTECTED

The **RealParentFeq** property contains the control number (field equate) of the parent control for window purposes. The `WebWindowClass` uses this property to allow child controls inherit properties from their parents.

This property is `PROTECTED`, therefore, it can only be referenced by a `WebControlClass` method, or a method in a class derived from `WebControlClass`.

Implementation:      The `Init` method sets the value of the `ParentFeq` property.

See Also:              `Init`

## WebControlClass Methods

The WebControlClass inherits all the methods of the HtmlItemClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebControlClass contains the methods listed below.

### Functional Organization—Expected Use

---

As an aid to understanding the WebControlClass, it is useful to organize the its various methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### Primary Interface Methods

The primary interface methods are called from outside the WebControlClass. That is, the WebWindowClass methods call these methods fairly routinely. These primary interface methods can be further divided into three categories according to their use:

##### **Housekeeping (one-time) Use:**

Init <sup>v</sup>	initialize WebControlClass object
SetAutoSpotLink <sup>v</sup>	generate live hypertext links
SetBorderWidth <sup>v</sup>	set control border width
SetQueue <sup>v</sup>	set list data source queue
Kill <sup>v</sup>	shut down WebControlClass object

##### **Mainstream Use:**

BeforeResetControl <sup>v</sup>	preprocess control
CreateHtml <sup>v</sup>	write HTML for control and children
CreateJsldata <sup>v</sup>	update Web page control
RefreshDisabled	update Web page control status
ResetFromQueue <sup>v</sup>	record changes to Server LIST queue
ResetControl <sup>v</sup>	update Server control
UpdateCopyUse	save copy of control contents
UpdateCopyChoice	save selected item number

<sup>v</sup> These methods are also Virtual.

**Occasional Use:**

CreateColorParameters	write Java applet color parameters
CreateCellHeader	begin HTML for control attributes
CreateCellFooter	end HTML for control attributes
DoSetChildDefaults	set nested child controls
GetChoiceChanged	return selection change
GetFont	add font information
GetLevel	return nesting level
GetNameAttribute	return HTML control name
GetQuotedText	return control text in quotes
GetUseChanged	return contents changed indicator
SetDirty	force list refresh

**Virtual Methods**

Typically you will not call these methods directly—the Primary Interface methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

BeforeResetControl	preprocess control
CreateCellContents	write HTML for control
CreateHtml	write HTML for control and children
CreateJsldata	update Web page control
GetBackgroundColor	return background color
GetCanDisable	return disable-ability flag
GetCellAttributes	return control attributes
GetEventAction	return browser action
GetHasHotkey	control text may contain &
GetIsChild	return family identity
GetPosition	get control coordinates
GetText	return control text
GetVisible	return control status flag
Init	initialize WebControlClass object
Kill	shut down WebControlClass object
PopFont	restore pre-PushFont font
PushFont	implement control font
ResetFromQueue	record changes to Server LIST queue
ResetControl	update server control
SetAutoSpotLink	generate live hypertext links
SetBorderWidth	set control border width
SetChildDefaults	set Web page control children
SetDirty	force list refresh
SetEventAction	associate browser action with control
SetParentDefaults	confirm parent control
SetQueue	set list data source queue

## BeforeResetControl (a virtual for control preprocessing)

---

### BeforeResetControl, VIRTUAL

The **BeforeResetControl** method is a virtual placeholder to do any control specific processing prior to synchronizing the window control with its corresponding Web page control.

BeforeResetControl is a VIRTUAL method so that other base class methods can directly call the BeforeResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

## CreateCellContents (write HTML for control)

---

### CreateCellContents( *html object* ), VIRTUAL, PROTECTED

#### CreateCellContents

A virtual placeholder to write the HTML code representing the control.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method is a virtual placeholder to write the HTML code representing the control.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method is PROTECTED, therefore, it can only be called by a WebControlClass method, or a method in a class derived from WebControlClass.

Implementation:

The proper positioning of each Web page control is accomplished by using HTML <TABLE> and <CELL> statements. CreateCellContents writes only the HTML code representing the control. The HTML code defining the <TABLE> and the <CELL> containing the control is written by other methods including LayoutHtmlClass.CreateHtml, CreateCellHeader, and CreateCellFooter.

See Also:

CreateHtml, CreateCellHeader, CreateCellFooter,  
LayoutHtmlClass.CreateHtml

## CreateCellFooter (end HTML for control attributes)

---

### CreateCellFooter( *html object* ), PROTECTED

**CreateCellFooter** Ends the HTML representing control attributes.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateCellFooter** method ends the HTML representing control attributes.

This method is PROTECTED, therefore, it can only be called by a WebControlClass method, or a method in a class derived from WebControlClass.

Implementation: CreateCellHeader ends the HTML Font specifications for the control.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CODE
SELF.CreateCellHeader(Target)
SELF.CreateCellContents(Target)
SELF.CreateCellFooter(Target)
```

## CreateCellHeader (begin HTML for control attributes)

---

### CreateCellHeader( *html object* ), PROTECTED

**CreateCellHeader** Begins the HTML representing control attributes.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateCellHeader** method begins the HTML representing control attributes.

This method is PROTECTED, therefore, it can only be called by a WebControlClass method, or a method in a class derived from WebControlClass.

Implementation: CreateCellHeader begins the HTML Font specifications for the control.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CODE
SELF.CreateCellHeader(Target)
SELF.CreateCellContents(Target)
SELF.CreateCellFooter(Target)
```

## CreateColorParameters (write Java applet color parameters)

**CreateColorParameters( *html object* [, *autospotlink*] ), PROTECTED**

### CreateColorParameters

Generates the HTML/JavaScript to specify the control's foreground and background colors.

*html object* The label of the HtmlClass object that writes the HTML code.

*autospotlink* An integer constant, variable, EQUATE, or expression indicating whether this control is a hyperlink. A value of one (1 or True) indicates a hyperlink; a value of zero (0 or False) indicates no hyperlink. If omitted, *autospotlink* defaults to False.

The **CreateColorParameters** method generates the HTML or JavaScript to specify the control's foreground and background colors.

This method is PROTECTED, therefore, it can only be called by a WebControlClass method, or a method in a class derived from WebControlClass.

#### Implementation:

The CreateColorParameters method sets the foreground color based on the CreateForeColorParameter method. It sets the background color based on the GetBackgroundColor method.

The autospotlink parameter has no effect, but is provided so colors can be adjusted for hyperlink controls.

#### Example:

```
WebJavaStringClass.CreateParams      PROCEDURE(*HtmlClass Target)
CurFont      HtmlFontClass
CODE
SELF.GetFont(CurFont)
SELF.LastText = SELF.GetText()
SELF.CreateColorParameters(Target, SELF.AutoSpotLink)
Target.WriteAppletOptParameter('Text', SELF.LastText)
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
Target.WriteAppletFontParameter(CurFont)
IF (SELF.CanBreak)
    Target.WriteAppletParameter('Wrap', '1')
END
IF (SELF.AutoSpotLink)
    Target.WriteAppletOptParameter('AutoSpotLink', SELF.AutoSpotLink)
END
```

#### See Also:

CreateForeColorParameter, GetBackgroundColor

## CreateForeColorParameter (write Java applet foreground color parameter)

**CreateForeColorParameter**( *html object* ), PROTECTED

### CreateForeColorParameter

Generates the HTML/JavaScript to specify the control's foreground color.

*html object*      The label of the HtmlClass object that writes the HTML code.

The **CreateForeColorParameter** method generates the HTML or JavaScript to specify the control's foreground color.

This method is PROTECTED, therefore, it can only be called by a WebControlClass method, or a method in a class derived from WebControlClass.

Implementation:      The CreateForeColorParameter method sets the foreground color based on the Clarion control's FONT attribute.

Example:

```
WebControlClass.CreateColorParameters PROCEDURE(*HtmlClass Target, BYTE AutoSpotLink)
BackColor                    LONG, AUTO
CODE
SELF.CreateForeColorParameter(Target)
BackColor = SELF.GetBackgroundColor()
IF (BackColor <> COLOR:None)
    Target.WriteAppletParameter('BackColor', IC:RGB(BackColor))
END
IF (AutoSpotLink)
END
```

## CreateHtml (write HTML for control and its attributes)

### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the control and its attributes.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the control and its attributes. CreateHtml also serves as a placeholder method for many derived classes—one for each control type. See *WebControlClass Derived Classes*.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The LayoutHtmlClass.CreateHtml method calls the WebControlClass.CreateHtml method.

Example:

```
MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem      &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)           !for all controls
  GET(SELF.Rows.Columns, Xindex)
  CurCell &= SELF.Rows.Columns.Cell
  NumItems = RECORDS(CurCell.Contents)
  Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
  LOOP Index = 1 TO NumItems                             !for all CELL items
    CurItem &= CurCell.GetItem(Index)                   !set WebControl object
    Style = CurItem.GetCellAttributes(Target)           !get control attributes
    IF (Style)
      Target.Write('<<P ' & Style & '>')               !begin HTML STYLE
    END
    CurItem.CreateHtml(Target)                           !write HTML control
    IF (Style)
      Target.Write('<</P>')                             !end HTML STYLE
    END
  END
  Target.WriteLine('<</TD>')                             !end HTML CELL
END
!procedure code
```

See Also: [LayoutHtmlClass.CreateHtml](#)

## CreateHtmlExtra (write HTML for related control)

---

### CreateHtmlExtra( *html object* ), VIRTUAL

**CreateHtmlExtra** A virtual placeholder to write any HTML required by a related control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtmlExtra** method is a virtual placeholder method to write HTML required by another, related control. For example, a WebHtmlRegionClass object may generate some HTML code required by its parent WebHtmlImageClass object.

CreateHtmlExtra is a VIRTUAL method so that other base class methods can directly call the CreateHtmlExtra virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: WebHtmlRegionClass.CreateHtmlExtra

## CreateJsldata (update Web page controls)

---

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Is a virtual placeholder method to write Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method is a virtual placeholder method to write Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the WebControlClass.CreateJsldata method.

## CreateParams (write all parameters for Java control)

---

### CreateParams( *html object* ), VIRTUAL

**CreateParams** A virtual placeholder to write all the applet parameters for a Java control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateParams** method is a virtual placeholder method to write all the applet parameters for a Java control.

CreateParams is a VIRTUAL method so that other base class methods can directly call the CreateParams virtual method in a derived class. This lets you easily implement your own custom version of this method.

## DoSetChildDefaults (set nested child controls)

---

### DoSetChildren, PROTECTED

The **DoSetChildDefaults** method sets the children of this control, considering only controls that have the same parent. This has the effect of properly relating layered (stacked) controls, nested controls, or both. For example a REGION control on top of an IMAGE control (layered to generated events for the IMAGE), or a SHEET control and its TAB controls on top of a LIST control (layered and nested to give the appearance of a LIST for each TAB), or a BUTTON control within an OPTION control (simple nesting to give proper visual organization of related controls).

This method is PROTECTED, therefore, it can only be called by a WebControlClass method, or a method in a class derived from WebControlClass.

Implementation: The DoSetChildDefaults method calls the WebWindowClass.GetChildren method to identify siblings, and the WebWindowClass.SetParentDefaults method to set the ParentFeq property for the children of this control.

Example:

```
WebHtmlSheetClass.SetChildDefaults PROCEDURE  
CODE  
PARENT.DoSetChildDefaults
```

See Also: ParentFeq, WebWindowClass.GetChildren,  
WebWindowClass.SetParentDefaults

## GetAlignText (return text alignment information)

---

### GetAlignText, STRING

The **GetAlignText** method returns a the control text alignment information for the WINDOW control, such as 'LEFT,' 'RIGHT,' or 'CENTER.' GetAlignText returns a null string if there is no alignment information.

Implementation: The GetAlignText method provides text alignment information for Java applets that support text alignment.

Return Data Type: STRING

Example:

```
WebJavaButtonClass.CreateHtml PROCEDURE(*HtmlClass Target,SIGNED Width,SIGNED Height)
Filename      CSTRING(FILE:MaxFileName)
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
SELF.IsEnabled = SELF.Feq{PROP:enabled}
Filename = SELF.GetFilename()
Target.WriteAppletHeader(SELF.Feq, 'ClarionImageButton', Width, Height)
Target.WriteAppletFilenameParameter('Picture', Filename)
Target.WriteAppletOptParameter('Label', SELF.GetText())
Target.WriteAppletOptParameter('Hint', SELF.Feq{PROP:tooltip})
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
IF (SELF.GetEventAction(EVENT:Accepted) = Update:Full)
    Target.WriteAppletParameter('Submit', 1)
END
IF (NOT SELF.IsEnabled)
    Target.WriteAppletOptParameter('Disabled', '1')
END
Target.WriteAppletFooter
```

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method is a virtual placeholder method to return the applet type for the control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

GetAppletType is a VIRTUAL method so that other base class methods can directly call the GetAppletType virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: STRING

## GetBackgroundColor (return background color)

---

**GetBackgroundColor( [default color] ), LONG, VIRTUAL**

### GetBackgroundColor

Returns the background color of the control.

*default color* An integer variable, constant, EQUATE, or expression containing the color to return if there is no background color. If omitted, *default color* defaults to Color:None.

The **GetBackgroundColor** method returns the background color of the control.

GetBackgroundColor is a VIRTUAL method so that other base class methods can directly call the GetBackgroundColor virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetBackgroundColor method returns the COLOR attribute of the window control, if there is one. Otherwise, it calls the GetParentBackgroundColor method to supply an inherited color.

EQUATEs for the *default color* parameter are declared in \LIBSRC\EQUATES.CLW.

Return Data Type: LONG

Example:

```
MyWebControlClass.CreateColorParameters PROCEDURE(*HtmlClass Target, BYTE AutoSpotLink)
ForeColor          LONG,AUTO
BackColor          LONG,AUTO
CODE
GETFONT(SELF.Feq,,, ForeColor)
BackColor = SELF.GetBackgroundColor()
IF (ForeColor <> 0) AND (ForeColor <> COLOR:None)
    Target.WriteAppletParameter('ForeColor', IC:RGB(ForeColor))
END
IF (BackColor <> COLOR:None)
    Target.WriteAppletParameter('BackColor', IC:RGB(BackColor))
END
```

See Also: GetParentBackgroundColor

## GetCanDisable (return disable-ability flag)

---

### GetCanDisable, BYTE, VIRTUAL

The **GetCanDisable** method returns a value indicating whether the Web page control can be disabled. A return value of one (1) indicates the control can be disabled; a return value of zero (0) indicates the control cannot be disabled.

GetCanDisable is a VIRTUAL method so that other base class methods can directly call the GetCanDisable virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebControlClass.GetVisible method calls the GetCanDisable method.

Return Data Type: BYTE

Example:

```
WebControlClass.GetVisible PROCEDURE  
  
CODE  
IF (SELF.DisabledAction = DISABLE:Hide) OR |  
  ((SELF.DisabledAction = DISABLE:OptHide) AND NOT SELF.GetCanDisable())  
  IF NOT SELF.Feq{PROP:enabled}  
    RETURN FALSE  
  END  
END  
RETURN SELF.Feq{PROP:visible}
```

See Also: WebControlClass.GetVisible, WebControlClass.DisabledAction

## GetCellAttributes (return control attributes)

**GetCellAttributes**( *html object* ), **STRING**, **VIRTUAL**

**GetCellAttributes** Returns HTML to set attributes associated with the Web page control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **GetCellAttributes** method returns HTML to set attributes associated with the Web page control. If there are no associated attributes, **GetCellAttributes** returns a null string.

**GetCellAttributes** is a **VIRTUAL** method so that other base class methods can directly call the **GetCellAttributes** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **GetCellAttributes** method returns HTML to set the font typeface, size, color, and style of the HTML `<TABLE> <CELL>` containing the control.

The `LayoutHtmlClass.CreateHtml` method calls the **GetCellAttributes** method.

Return Data Type: **STRING**

Example:

```
MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)

CurCell      &LayoutCellClass,AUTO
Index         SIGNED,AUTO
NumRows       SIGNED,AUTO
Xindex        SIGNED,AUTO
Yindex        SIGNED,AUTO

CODE
NumRows = RECORDS(SELF.Rows)
Target.Write('<<TABLE' & SELF.Style & '>')
LOOP Yindex = 1 TO NumRows
  GET(SELF.Rows, Yindex)
  Target.Write('<<TR>')
  LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)
    GET(SELF.Rows.Columns, Xindex)
    CurCell &= SELF.Rows.Columns.Cell
    Target.Write('<<TD' & CurCell.GetCellAttributes() & '>')
    Target.WriteLine('<</TD>')
  END
  Target.Write('<</TR>')
END
Target.WriteLine('<</TABLE>')
```

See Also: `LayoutHtmlClass.CreateHtml`

## GetChoiceChanged (return selection change)

---

### GetChoiceChanged, BYTE

The **GetChoiceChanged** method returns a value indicating whether the end user selected a different item within the control. A return value of one (1) indicates a new selection; a return value of zero (0) indicates no change.

The WebControlClass uses this method and the UpdateCopyChoice method to minimize the data sent to the Client browser—no Java Support Library (JSL) data is generated for unchanged controls.

Return Data Type:       **BYTE**

Example:

```
WebHtm1OptionClass.CreateJs1Data     PROCEDURE(*Js1ManagerClass Target)

CODE
IF (SELF.GetChoiceChanged())
    Target.SetValue(SELF.Feq, CHOICE(SELF.Feq))
    SELF.UpdateCopyChoice
END
```

See Also:               **UpdateCopyChoice**

## GetEventAction (return browser action)

### GetEventAction( *event* ), SIGNED, VIRTUAL

**GetEventAction** Returns the browser action for the specified *event*.

*event* An integer constant, variable, EQUATE, or expression indicating the event for which to return the action.

The **GetEventAction** method returns the browser action associated with the specified *event*. The WebControlClass object uses this method to generate appropriate HTML/JavaScript for the control.

The SetEventAction associates browser actions with events.

GetEventAction is a VIRTUAL method so that other base class methods can directly call the GetEventAction virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: EQUATES for the return values are declared in ICSTD.EQU as follows:

EVENT:NewPage	EQUATE(280H)	! Build a new page now
EVENT:Initialize	EQUATE(281H)	! List box wants contents
EVENT:RefreshPage	EQUATE(282H)	! Refresh a browse box
Update:OnBrowser	EQUATE(0)	
Update:Partial	EQUATE(1)	
Update:Full	EQUATE(2)	

Return Data Type: **SIGNED**

Example:

```
WebJavaButtonClass.CreateParams PROCEDURE(*HtmlClass Target)
Filename CSTRING(FILE:MaxFileName)
CODE
SELF.IsEnabled = SELF.Feq{PROP:enabled}
Filename = SELF.GetFilename()
Target.WriteAppletFilenameParameter('Picture', Filename)
Target.WriteAppletOptParameter('Label', SELF.GetText())
Target.WriteAppletOptParameter('Hint', SELF.Feq{PROP:tooltip})
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
IF (SELF.GetEventAction(EVENT:Accepted) = Update:Full)
    Target.WriteAppletParameter('Submit', 1)
END
IF (NOT SELF.IsEnabled)
    Target.WriteAppletOptParameter('Disabled', '1')
END
```

See Also: **SetEventAction**

## GetFont (add font information)

### GetFont( *font object* )

**GetFont** Adds the control's font information to the *font object*.  
*font object* The label of the HtmlFontClass object that manages font information.

The **GetFont** method adds the control's font information to the *font object* for subsequent retrieval by other methods such as PushFont. The font information includes the typeface, size, color, and style.

Example:

```
WebHtmlButtonClass.CreateCellContents PROCEDURE(*HtmlClass Target)
Border          SIGNED(0)
CODE
Target.Write('<<INPUT TYPE=SUBMIT VALUE=""')
Target.Write(IC:QuoteText(SELF.GetText(), IC:RESET:HotValue) & '""')
Target.Write(SELF.GetNameAttribute(Target))
Target.WriteLine('>')
```

See Also:           PushFont

## GetHasHotkey (control text contains '&')

### GetHasHotKey, BYTE, VIRTUAL

The **GetHasHotKey** method returns a value indicating whether the control text may contain the Clarion hot key delimiter (&). A return value of one (1) indicates a possible hot key; a return value of zero (0) indicates no hot key. The WebControlClass object uses this method to remove the hot key delimiter from Web page text.

GetHasHotKey is a VIRTUAL method so that other base class methods can directly call the GetHasHotKey virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:   The GetText method calls the GetHasHotKey method to identify controls that may have Clarion hot key delimiters (&).

Return Data Type:    BYTE

Example:

```
WebControlClass.GetText PROCEDURE
CODE
IF (SELF.GetHasHotkey())
RETURN IC:StripHotkey(CLIP(SELF.Feq{PROP:ScreenText}))
END
RETURN CLIP(SELF.Feq{PROP:ScreenText})
```

See Also:           GetText

## GetIsChild (return family identity)

**GetIsChild( *parent control* [, *control type*] ), SIGNED, VIRTUAL**

<b>GetIsChild</b>	Returns a value indicating whether this WebControlClass object is a visible child control of the specified <i>parent control</i> and <i>control type</i> .
<i>parent control</i>	An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.
<i>control type</i>	An integer constant, variable, EQUATE, or expression containing the type of control sought. If omitted, any control type is valid.

The **GetIsChild** method returns a value indicating whether this WebControlClass object is a visible child control of the specified *parent control* and *control type*. If this WebControlClass object meets the specified criteria it returns its own control number (Feq property), otherwise it returns zero (0) to indicate it does not meet the specified criteria.

GetIsChild is a VIRTUAL method so that other base class methods can directly call the GetIsChild virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.GetFirstChild method calls the GetIsChild method to find first child controls.

EQUATES for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE:.

Return Data Type: SIGNED

Example:

```
WebWindowClass.GetFirstChild      PROCEDURE(SIGNED ParentFeq, SIGNED Type=0)
Index          SIGNED,AUTO
FirstChild     SIGNED(0)
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.ParentFeq = ParentFeq)
    FirstChild = SELF.Controls.ThisControl.GetIsChild(ParentFeq, Type)
    IF (FirstChild)
      BREAK
    END
  END
END
END
RETURN FirstChild
```

See Also: Feq, WebWindowClass.GetFirstChild

## GetLevel (return nesting level)

---

### GetLevel, SIGNED

The **GetLevel** method returns the control's nesting level. The **WebControlClass** object uses this method to properly position (indent) menus and menu items.

Return Data Type:      **SIGNED**

Example:

```
WebHtmlMenuClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE
Target.WriteLine('<<NOBR>')
Target.WriteSpace((SELF.GetLevel()-1)*2)
Target.WriteLine(SELF.GetQuotedText())
Target.WriteLine('<</NOBR><<BR>')
```

## GetNameAttribute (return HTML control name)

---

### GetNameAttribute( *html object* ), STRING

**GetNameAttribute** Returns the HTML name of the Web page control.

*html object*            The label of the **HtmlClass** object that writes the HTML code.

The **GetNameAttribute** method returns the HTML name of the Web page control. The **WebControlClass** object uses this method to generate the control name in HTML code.

Implementation:        The **GetNameAttribute** method converts the Clarion control number (field equate) into a valid, unique HTML control name.

Return Data Type:      **STRING**

Example:

```
WebHtmlButtonClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
Border                    SIGNED(0)
```

```
CODE
Target.Write('<<INPUT TYPE=SUBMIT VALUE=""')
Target.Write(IC:QuoteText(SELF.GetText(), IC:RESET:HotValue) & '""')
Target.Write(SELF.GetNameAttribute(Target))
Target.WriteLine('>')
```

## GetParentBackgroundColor (return parent background color)

---

**GetParentBackgroundColor**( [*default color*] ), LONG, VIRTUAL

### **GetParentBackgroundColor**

Returns the background color of the parent control.

*default color* An integer variable, constant, EQUATE, or expression containing the color to return if there is no background color. If omitted, *default color* defaults to Color:None.

The **GetParentBackgroundColor** method returns the background color of the parent control or window. The WebControlClass object uses this method to allow controls to inherit colors from their parent controls.

GetParentBackgroundColor is a VIRTUAL method so that other base class methods can directly call the GetParentBackgroundColor virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetParentBackgroundColor method calls the GetBackgroundColor method for the control's parent. The parent may be another control or the WINDOW.

EQUATEs for the *default color* parameter are declared in \LIBSRC\EQUATES.CLW.

Return Data Type: LONG

Example:

```
WebControlClass.GetBackgroundColor PROCEDURE
BackColor LONG
CODE
BackColor = SELF.Feq{PROP:background}
IF (BackColor <> COLOR:None)
RETURN BackColor
END
RETURN SELF.GetParentBackgroundColor()
```

See Also: GetBackgroundColor

## GetPosition (get control coordinates)

**GetPosition**( *x, y, width, height* ), VIRTUAL

<b>GetPosition</b>	Returns the control coordinates.
<i>x</i>	A numeric variable to receive the control's horizontal position.
<i>y</i>	A numeric variable to receive the control's vertical position.
<i>width</i>	A numeric variable to receive the control's width.
<i>height</i>	A numeric variable to receive the control's height.

The **GetPosition** method returns the control's WINDOW coordinates. The WebWindowClass and the WebControlClass use this method to help set appropriate position-based parent/child relationships for the controls. The LayoutHtmlClass uses this information to help position the control on the Web page.

GetPosition is a VIRTUAL method so that other base class methods can directly call the GetPosition virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetPosition method uses the GETPOSITION procedure. See the *Language Reference* for more information.

Example:

```
WebControlClass.DoSetChildDefaults PROCEDURE
Children    WebControlQueue
MyRect      GROUP(Rect)
            END
CODE
SELF.GetPosition(MyRect.x, MyRect.y, MyRect.width, MyRect.height)
SELF.OwnerWindow.GetChildren(Children, SELF.ParentFeg)
SELF.OwnerWindow.SetParentDefaults(Children, SELF, MyRect)
```

## GetQuotedText (return control text in quotes)

---

### GetQuotedText, STRING

The **GetQuotedText** method returns control text within quotation marks recognized by the Client browser. For example, some browsers require single quotes ('), while others require double quotes (").

Implementation: **GetQuotedText** calls the **GetText** method to retrieve the control's text.

Return Data Type: **STRING**

Example:

```
WebHtmlItemClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE
```

```
Target.WriteLine('<<NOBR>')
```

```
Target.WriteSpace((SELF.GetLevel()-1)*2)
```

```
Target.Write('<<A HREF="' & Target.GetControlReference(SELF.Feq) & '">')
```

```
Target.WriteLine(SELF.GetQuotedText() & '<</A>')
```

```
Target.WriteLine('<</NOBR><<BR>')
```

See Also: **GetText**

## GetTableAttributes (return HTML STYLE)

---

### GetTableAttributes, STRING, VIRTUAL

The **GetTableAttributes** method returns the HTML **<STYLE>** string for the HTML **<TABLE>** representing the control. Several **WebControlClass** objects, such as **WebHtmlGroupClass** and **WebHtmlOptionClass** objects translate their Clarion controls to an HTML **<TABLE>**.

**GetTableAttributes** is a **VIRTUAL** method so that other base class methods can directly call the **GetTableAttributes** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **<STYLE>** returned by **GetTableAttributes** includes only the **<TABLE>** **<BORDER>**.

Return Data Type: **STRING**

Example:

```
WebHtmlOptionClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
```

```
SELF.OwnerWindow.CreateChildHtml(Target, SELF.Feq, SELF.GetTableAttributes())
```

```
SELF.UpdateCopyChoice
```

## GetText (return control text)

---

**GetText, STRING, VIRTUAL**

The **GetText** method returns the control text minus any Clarion hot key delimiter (&).

GetText is a VIRTUAL method so that other base class methods can directly call the GetText virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetQuotedText method calls the GetText method to retrieve the text. GetText calls the GetHasHotKey method to identify controls that may have Clarion hot key delimiters (&). GetText returns PROP:ScreenText for the control.

Return Data Type: STRING

Example:

```
WebControlClass.GetQuotedText      PROCEDURE
CODE
RETURN IC:QuoteText(SELF.GetText(), IC:RESET:Text)
```

See Also: GetHasHotKey, GetQuotedText

**GetUseChanged (return contents changed indicator)****GetUseChanged, BYTE**

The **GetUseChanged** method returns a value indicating whether the control contents changed. A return value of one (1) indicates a change; a return value of zero (0) indicates no change. The WebControlClass uses this method and the UpdateCopyUse method to minimize the data sent to the Client browser—no JSL data is generated for unchanged controls.

Return Data Type: BYTE

Example:

```
WebHtmlTextClass.CreateJsldata      PROCEDURE(*Js1ManagerClass Target)
CODE
IF (SELF.GetUseChanged())
    Target.SetValue(SELF.Feq, CONTENTS(SELF.Feq))
    SELF.UpdateCopyUse
END
```

See Also: UpdateCopyUse

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the control should appear on the Web page. A return value of one (1) indicates the control should appear on the Web page; a return value of zero (0) indicates the control should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

**Implementation:** A control may be omitted from the Web page for several reasons: it may be disabled or hidden, it's parent may be disabled or hidden, or it may be on a TAB control that is not selected.

**Return Data Type:** BYTE

**Example:**

```
WebWindowClass.CreateJsldata  PROCEDURE(*Js1ManagerClass Target)
Index                          SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.GetVisible())
    SELF.Controls.ThisControl.CreateJsldata(Target)
  END
END
END
```

## Init (initialize the WebControlClass object)

---

**Init**( *control number*, *owner window* ), **VIRTUAL**

<b>Init</b>	Initializes the WebControlClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate).
<i>owner window</i>	The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebControlClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The Init method sets the initial value of the ActionOnAccept, DisabledAction, Feq, OwnerWindow, and ParentFeq properties.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)

CODE
ASSERT(~NewControl &= NULL)
NewControl.Init(Feq, SELF)
SELF.AddControl(NewControl)
```

See Also: ActionOnAccept, DisabledAction, Feq, OwnerWindow, ParentFeq, WebWindowClass.AddControl

## Kill (shut down the WebControlClass object)

---

### Kill, VIRTUAL

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Kill is a VIRTUAL method so that other base class methods can directly call the Kill virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebControlClass.Kill method does nothing. It is a placeholder for derived class methods such as WebAreaClass.Kill and WebJavaListClass.Kill.

The WebWindowClass.Kill method calls the Kill method.

Example:

```
MyWebWindowClass.Kill          PROCEDURE
Index          SIGNED,AUTO
CODE
IF (~SELF.Controls &= NULL)
  LOOP Index = 1 TO RECORDS(SELF.Controls)
    GET(SELF.Controls, Index)
    ASSERT(~SELF.Controls.ThisControl &= NULL)
    SELF.Controls.ThisControl.Kill
  END
DISPOSE(SELF.Controls)
END
```

See Also: WebAreaClass.Kill, WebJavaListClass.Kill.

## PopFont (restore pre-PushFont font)

---

### PopFont( *html object* ), VIRTUAL

<b>PopFont</b>	Restores font information to its pre-PushFont state.
<i>html object</i>	The label of the HtmlClass object that writes the HTML code.

The **PopFont** method restores font information to its pre-PushFont state. Appropriate use of PushFont and PopFont allow child controls to correctly inherit or override parent control font information. Appropriate use simply means pairing a call to PushFont with a subsequent call to PopFont.

PopFont is a VIRTUAL method so that other base class methods can directly call the PopFont virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The PopFont method calls the HtmlClass.PopFont method.

Example:

```
WebMenubarClass.CreateHtml       PROCEDURE(*HtmlClass Target)

CODE
SELF.PushFont(Target)
SELF.OwnerWindow.CreateChildHtml(Target, FEQ:Menubar)
SELF.PopFont(Target)
```

See Also:               **PushFont**

## PushFont (implement control font)

---

### PushFont( *html object* ), VIRTUAL

**PushFont** Implements the control's font in the HTML generated for the control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **PushFont** method implements the control's font in the HTML generated for the control. Appropriate use of PushFont and PopFont allow child controls to correctly inherit or override parent control font information. Appropriate use simply means pairing a call to PushFont with a subsequent call to PopFont.

PushFont is a VIRTUAL method so that other base class methods can directly call the PopFont virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The PopFont method calls the HtmlClass.PopFont method.

Example:

```
WebMenuBarClass.CreateHtml      PROCEDURE(*HtmlClass Target)

CODE
SELF.PushFont(Target)
SELF.OwnerWindow.CreateChildHtml(Target, FEQ:MenuBar)
SELF.PopFont(Target)
```

See Also: PopFont

## RefreshDisabled (update Web page control status)

---

### RefreshDisabled( *Jsl manager, Server enabled, Client enabled* )

<b>RefreshDisabled</b>	Updates the control's enabled/disabled state.
<i>Jsl manager</i>	The label of the JSLManagerClass object that sends data to Java applets on the Web page.
<i>Server enabled</i>	A Boolean constant, variable, EQUATE, or expression indicating the state of the control in the Web-enabled Clarion application. A value of one (1) indicates an enabled control; a zero (0) indicates a disabled control.
<i>Client enabled</i>	A Boolean constant, variable, EQUATE, or expression indicating the state of the control on the browser Web page. A value of one (1) indicates an enabled control; a zero (0) indicates a disabled control.

The **RefreshDisabled** method applies the Server control's enabled/disabled state to the corresponding Web page control. The WebControlClass uses this method to make sure the Web page control is enabled or disabled concurrently with its corresponding Windows control.

Example:

```
WebHtmlTabClass.CreatedJsldata    PROCEDURE(*JslManagerClass Target)

CODE
Target.SelectControl(SELF.Feq)
SELF.RefreshDisabled(Target, 1 - SELF.Feq{PROP:disable}, SELF.IsEnabled)
```

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server control with its corresponding Web page control.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server control with its corresponding Web page control. The **ResetControl** method takes Web page control information submitted by the Client browser, such as control contents and events, and applies it to the corresponding Server control.

**ResetControl** is a VIRTUAL method so that other base class methods can directly call the **ResetControl** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebWindowClass.TakeRequest` method calls the **ResetControl** method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit    &SubmitItemClass
CurFeq       SIGNED,AUTO
CODE
LOOP
  NextSubmit &= SELF.Server.SetNextAction()
  IF (NextSubmit &= NULL)
    BREAK
  END
  CurFeq = NextSubmit.Feq
  IF (SELF.GetControl(CurFeq))
    IF (NextSubmit.Event)
      IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
        SELF.Controls.ThisControl.ResetControl(NextSubmit)
      END
    END
  END
END
END
END
```

See Also: `WebWindowClass.TakeRequest`

## ResetFromQueue (record changes to Server LIST queue)

**ResetFromQueue**( *change* [,*offset*] [,*number*] ), VIRTUAL

<b>ResetFromQueue</b>	A virtual to record changes to the Server LIST control.
<i>change</i>	An integer constant, variable, EQUATE or expression that indicates the type of change made to the LIST's queue. Valid actions include insert, delete, delete all, replace, scroll, scroll down, and scroll up.
<i>offset</i>	An integer constant, variable, EQUATE or expression that indicates the direction of a scroll action, or the relative position of an insert, delete, or replace action. If omitted, <i>offset</i> defaults to zero (0).
<i>number</i>	An integer constant, variable, EQUATE or expression that indicates the number of list items to scroll for a scroll action; otherwise the number of list items affected for an insert, delete, or replace action. If omitted, <i>number</i> defaults to one (1).

The **ResetFromQueue** method is a virtual placeholder method to record changes to the Server LIST control (its FROM attribute or data source queue), so the same changes can be efficiently applied to the corresponding Web page control.

ResetFromQueue is a VIRTUAL method so that other base class methods can directly call the ResetFromQueue virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

EQUATEs for the how parameter are declared in ICCNTRLS.INC as follows:

```

ITEMIZE, PRE(ACTION)
Insert      EQUATE
Delete      EQUATE
Replace     EQUATE
DeleteAll   EQUATE
Scroll      EQUATE
ScrollDown  EQUATE
ScrollUp    EQUATE
END

```

See Also:

WebJavaListClass.ResetFromQueue

## SetAutoSpotLink (a virtual to set AutoSpotLink)

---

### SetAutoSpotLink( *Boolean* ), VIRTUAL

**SetAutoSpotLink** A virtual placeholder method to set the value of the AutoSpotLink property in derived classes.

*Boolean* A Boolean constant, variable, EQUATE or expression assigned to the AutoSpotLink property. A value of one (1) generates live hypertext links; a value of zero (0) generates plain text.

The **SetAutoSpotLink** method is a virtual placeholder method to set the value of the AutoSpotLink property in derived classes. The AutoSpotLink property indicates whether control text that appears to be a URL, email address, FTP site, etc. is rendered on the Web page as a live hypertext link.

SetAutoSpotLink is a VIRTUAL method so that other base class methods can directly call the SetAutoSpotLink virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: [WebJavaListClass.SetAutoSpotLink](#), [WebJavaStringClass.SetAutoSpotLink](#)

## SetBorderWidth (a virtual to set BorderWidth)

---

### SetBorderWidth( *width* ), VIRTUAL

**SetBorderWidth** A virtual placeholder method to set the value of the BorderWidth property in derived classes.

*width* An integer constant, variable, EQUATE or expression assigned to the BorderWidth property.

The **SetBorderWidth** method is a virtual placeholder method to set the value of the BorderWidth property in derived classes. The BorderWidth property determines the Web page control's border width.

SetBorderWidth is a VIRTUAL method so that other base class methods can directly call the SetBorderWidth virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: [WebHtmlGroupClass.SetBorderWidth](#),  
[WebHtmlOptionClass.SetBorderWidth](#), [WebHtmlSheetClass.SetBorderWidth](#)

## SetBreakable (allow word wrap)

---

### SetBreakable( *value* ), VIRTUAL

**SetBreakable** A virtual placeholder method to set the value of the CanBreak property in derived classes.

*value* An integer constant, variable, EQUATE or expression assigned to the CanBreak property.

The **SetBreakable** method is a virtual placeholder method to set the value of the CanBreak property in derived classes. The CanBreak property determines whether or not the control's text may be written on more than one line (wrap).

SetBreakable is a VIRTUAL method so that other base class methods can directly call the SetBreakable virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: [WebStringClass.SetBreakable](#)

## SetChildDefaults (a virtual to set Web page control children)

---

### SetChildDefaults, VIRTUAL

The **SetChildDefaults** method is a virtual placeholder method to set the children of the Web page control.

For Web page purposes, a parent/child relationship between controls is defined by the controls' positional coordinates. A control whose boundaries fall entirely within the boundaries of another control is the child of the surrounding control (if there is no other surrounding control). This position based relationship allows for correct positioning, alignment and processing of Web page controls.

SetChildDefaults is a VIRTUAL method so that other base class methods can directly call the SetChildDefaults virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: [WebHtmlImageClass.SetChildDefaults](#),  
[WebHtmlSheetClass.SetChildDefaults](#)

## SetDescription (a virtual to set AltText)

---

### SetDescription( *text* ), VIRTUAL

**SetDescription** A virtual placeholder method to set the value of the AltText property in derived classes.

*text* A string constant, variable, EQUATE or expression assigned to the AltText property.

The **SetDescription** method is a virtual placeholder method to set the value of the AltText property in derived classes.

SetDescription is a VIRTUAL method so that other base class methods can directly call the SetDescription virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: [WebHtmlImageClass.SetDescription](#)

## SetEventAction (associate browser action with control event)

### SetEventAction( *event*, *action* ), VIRTUAL

<b>SetEventAction</b>	Associates a browser action with a particular control event.
<i>event</i>	An integer constant, variable, EQUATE or expression identifying the Web control event. Most controls only support EVENT:Accepted, which is triggered when the end user presses an OK or Submit button, or the RETURN, ENTER, or TAB key. However, some list based controls also support scrolling events, selection events, and others.
<i>action</i>	An integer constant, variable, EQUATE or expression identifying the browser action. Valid actions are Update:OnBrowser (don't contact the Server ), Update:Partial (request Java control data only from the Server), and Update:Full (request the entire page from the Server).

The **SetEventAction** method associates a browser action with a particular Web page control event. That is, the browser action, such as Update:Partial, is an attribute of a control. When the end user creates an event for the Web page control by mouse-clicking or pressing the RETURN, ENTER, or TAB key, the Client browser requests the action associated with the event. The Clarion Application Broker forwards the request to the Server application which processes it.

SetEventAction is a VIRTUAL method so that other base class methods can directly call the SetEventAction virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

EQUATES for the *event* parameter are declared in EQUATES.CLW. Each event EQUATE is prefixed with EVENT:. Valid events are

```
EVENT:Accepted
EVENT:Initialize
EVENT:NewSelection
EVENT:AlertKey
EVENT:Locate
EVENT:ScrollTop
EVENT:ScrollBottom
EVENT:PageUp
EVENT:PageDown
EVENT:ScrollUp
EVENT:ScrollDown
EVENT:ScrollDrag
EVENT:Expanding
EVENT:Contracting
EVENT:Expanded
EVENT:Contracted
```

EQUATES for the *action* parameter are declared in ICSTD.EQU as follows:

```
Update:OnBrowser    EQUATE(0)
Update:Partial      EQUATE(1)
Update:Full         EQUATE(2)
```

Example:

```
WebJavaListClass.Init PROCEDURE(SIGNED Feq, *WebWindowBaseClass OwnerWindow)
```

```
CODE
PARENT.Init(Feq, OwnerWindow)
SELF.EventActionQ &= NEW EventActionQueue
SELF.QueueActionQ &= NEW QueueActionQueue
SELF.SendIcons = TRUE

IF (SELF.feq{PROP:imm})
    SELF.SetEventAction(EVENT:ScrollDown,    Update:Partial)
    SELF.SetEventAction(EVENT:ScrollUp,      Update:Partial)
    SELF.SetEventAction(EVENT:ScrollTop,     Update:Partial)
    SELF.SetEventAction(EVENT:ScrollBottom,  Update:Partial)
    SELF.SetEventAction(EVENT:ScrollDrag,    Update:Partial)
    SELF.SetEventAction(EVENT:PageUp,        Update:Partial)
    SELF.SetEventAction(EVENT:PageDown,      Update:Partial)
    SELF.SetEventAction(EVENT:Locate,        Update:Partial)
END

SELF.SetEventAction(EVENT:NewSelection,    Update:Partial)
SELF.SetEventAction(EVENT:Initialize,      Update:Partial)
SELF.SetEventAction(EVENT:AlertKey,        Update:Partial)
```

See Also: **ActionOnAccept, GetEventAction**

## SetParentDefaults (confirm parent)

**SetParentDefaults**( *potential parent*, *coordinates* ), **VIRTUAL**

**SetParentDefaults** Confirms this control's boundaries fall entirely within the *potential parent* boundaries.

*potential parent* The label of a WebControlClass object.

*coordinates* The label of a structure containing the coordinates of the *potential parent*.

The **SetParentDefaults** method confirms this control's boundaries fall entirely within the *potential parent* boundaries, and if so, sets the ParentFeq property to reflect the parent/child relationship.

The WebControlClass uses this method to establish the correct relationships between IMAGE and REGION controls and SHEET and LIST controls in template generated Browse procedures.

For Web page purposes, a parent/child relationship between controls is defined by the controls' positional coordinates. A control whose boundaries fall entirely within the boundaries of another control is the child of the surrounding control (if there is no other surrounding control). This position based relationship allows for correct positioning, alignment and processing of Web page controls.

SetParentDefaults is a VIRTUAL method so that other base class methods can directly call the SetParentDefaults virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The *coordinates* parameter names a GROUP with the same structure as the Rect GROUP declared in ICWINDOW.INC as follows:

```
Rect          GROUP,TYPE
x             SIGNED
y             SIGNED
width        SIGNED
height       SIGNED
END
```

Example:

```
WebControlListClass.SetParentDefaults PROCEDURE|
  (*WebControlQueue Source, *WebControlClass Other, *Rect ParentPos)

CurIndex          SIGNED,AUTO
CODE
LOOP CurIndex = 1 TO RECORDS(Source)
  GET(Source, CurIndex)
  Source.ThisControl.SetParentDefaults(Other, ParentPos)
END
```

See Also:

WebWindowClass.SetParentDefaults

## SetQueue (a virtual to set the FromQ property)

---

### SetQueue( *source* ), VIRTUAL

**SetQueue** Is a virtual placeholder method to set the value of the FromQ property in derived classes.

*source* The label of the data source QUEUE.

The **SetQueue** method is a virtual placeholder method to set the value of the FromQ property in derived classes. The FromQ property is a reference to the data source for the LIST control.

SetQueue is a VIRTUAL method so that other base class methods can directly call the SetQueue virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: [WebJavaListClass.FromQ](#), [WebJavaListClass.SetQueue](#)

## UpdateCopyChoice (save selected item number)

---

### UpdateCopyChoice, PROTECTED

The **UpdateCopyChoice** method saves the number of the Clarion control's currently selected item. The WebControlClass uses this information and the GetChoiceChanged method to minimize the data sent to the Client browser—no JSL data is generated for unchanged controls.

This method is PROTECTED, therefore, it can only be called by a WebControlClass method, or a method in a class derived from WebControlClass.

Example:

```
WebHtm1OptionClass.CreateJs1Data    PROCEDURE(*Js1ManagerClass Target)

CODE
IF (SELF.GetChoiceChanged())
    Target.SetValue(SELF.Feq, CHOICE(SELF.Feq))
    SELF.UpdateCopyChoice
END
```

See Also: [GetChoiceChanged](#)

## UpdateCopyUse (save copy of control contents)

---

### UpdateCopyUse, PROTECTED

The **UpdateCopyUse** method saves a copy of the Clarion control's contents. The `WebControlClass` uses this information and the `GetUseChanged` method to minimize the data sent to the Client browser—no JSL data is generated for unchanged controls.

This method is **PROTECTED**, therefore, it can only be called by a `WebControlClass` method, or a method in a class derived from `WebControlClass`.

Example:

```
WebHtmlTextClass.CreateJsldata      PROCEDURE(*Js1ManagerClass Target)

CODE
IF (SELF.GetUseChanged())
    Target.SetValue(SELF.Feq, CONTENTS(SELF.Feq))
    SELF.UpdateCopyUse
END
```

See Also:           **GetUseChanged**

# WEBCONTROLCLASS DERIVED CLASSES

<b>Overview</b>	<b>269</b>
WebControlClass Concepts .....	269
Classes Derived from WebControlClass .....	269
Relationship to Other Internet Builder Classes .....	269
Internet Connect Template Implementation .....	269
Source Files .....	270
<b>WebHtmlCheckClass Methods</b>	<b>271</b>
BeforeResetControl (control preprocessing) .....	271
CreateCellContents (generate HTML check box) .....	272
CreateJsldata (update Web page control) .....	273
GetHasHotkey (control text may contain &) .....	274
ResetControl (update server control) .....	275
<b>WebHtmlEntryClass Methods</b>	<b>276</b>
CreateCellContents (generate HTML entry box) .....	276
CreateJsldata (update Web page control) .....	277
ResetControl (update server control) .....	278
<b>WebHtmlGroupClass Properties</b>	<b>279</b>
BorderWidth (Web page control border width) .....	279
<b>WebHtmlGroupClass Methods</b>	<b>280</b>
CreateHtml (write HTML for GROUP control) .....	280
GetHasHotkey (control text may contain &) .....	281
Init (initialize WebHtmlGroupClass object) .....	282
SetBorderWidth (set Web page control border width) .....	283
<b>WebHtmlItemClass Methods</b>	<b>284</b>
CreateCellContents (generate HTML menu item) .....	284
GetVisible (return control status flag) .....	285
ResetControl (update server control) .....	286
<b>WebHtmlMenuClass Methods</b>	<b>287</b>
CreateCellContents (generate HTML menu) .....	287
CreateHtml (write HTML for MENU control) .....	288
GetCellAttributes (return control attributes) .....	289
GetVisible (return control status flag) .....	290

<b>WebHtmlOptionClass Properties</b>	<b>291</b>
BorderWidth (Web page control border width) .....	291
<b>WebHtmlOptionClass Methods</b>	<b>292</b>
CreateHtml (write HTML for OPTION control) .....	292
CreateJsldata (update Web page control) .....	293
GetTableAttributes (return HTML STYLE).....	294
Init (initialize WebHtmlOptionClass object) .....	295
ResetControl (update server control).....	296
SetBorderWidth (set Web page control border width) .....	297
<b>WebHtmlPromptClass Methods</b>	<b>298</b>
CreateCellContents (generate HTML prompt).....	298
GetHasHotkey (control text may contain &).....	299
<b>WebHtmlRadioClass Methods</b>	<b>300</b>
CreateCellContents (generate HTML radio button).....	300
GetHasHotkey (control text may contain &).....	301
<b>WebHtmlRegionClass Methods</b>	<b>302</b>
CreateHtmlExtra (write HTML for related control) .....	302
SetParentDefaults (confirm parent) .....	303
<b>WebHtmlSheetClass Properties</b>	<b>304</b>
BorderWidth (Web page control border width).....	304
<b>WebHtmlSheetClass Methods</b>	<b>305</b>
CreateHtml (write HTML for SHEETcontrol).....	305
CreateTabControl (write HTML for SHEET TABs) .....	307
GetIsChild (return family identity).....	308
GetTableAttributes (return HTML STYLE).....	309
Init (initialize WebHtmlSheetClass object) .....	310
ResetControl (update server control).....	311
SetBorderWidth (set Web page control border width) .....	312
SetChildDefaults (set nested child controls) .....	313
<b>WebHtmlTabClass Properties</b>	<b>314</b>
IsEnabled (control enabled flag) .....	314
<b>WebHtmlTabClass Methods</b>	<b>315</b>
CreateHtml (write HTML for control and its attributes).....	315
CreateJsldata (update Web page control) .....	316
CreateParams (write all tab parameters) .....	317
GetAppletType (return applet type).....	318
GetHasHotkey (control text may contain &).....	319

GetIsChild (return family identity) .....	320
GetPosition (get control coordinates) .....	321
GetVisible (return control status flag) .....	322
ResetControl (update server control) .....	323
SetParentDefaults (confirm parent) .....	324
<b>WebHtmlTextClass Methods</b> .....	<b>325</b>
CreateCellContents (generate HTML text box) .....	325
CreateJslData (update Web page control) .....	326
ResetControl (update server control) .....	327
<b>WebButtonClass Methods</b> .....	<b>328</b>
BeforeResetControl (control preprocessing) .....	328
GetHasHotkey (control text may contain &) .....	329
GetVisible (return control status flag) .....	330
ResetControl (update server control) .....	331
<b>WebHtmlButtonClass Methods</b> .....	<b>332</b>
CreateCellContents (generate HTML button) .....	332
Init (initialize WebButtonClass object) .....	333
ResetControl (update server control) .....	334
<b>WebJavaButtonClass Properties</b> .....	<b>335</b>
IsEnabled (control enabled flag) .....	335
<b>WebJavaButtonClass Methods</b> .....	<b>336</b>
CreateHtml (write HTML for control and its attributes) .....	336
CreateJslData (update Web page control) .....	338
CreateParams (write all button parameters) .....	339
GetAppletType (return applet type) .....	339
GetCanDisable (return disable-ability flag) .....	340
GetFilename (return image filename) .....	341
ResetControl (update server control) .....	342
<b>WebJavaToolButtonClass Methods</b> .....	<b>343</b>
GetEventAction (return browser action) .....	343
<b>WebImageClass</b> .....	<b>345</b>
<b>WebHtmlImageClass Properties</b> .....	<b>346</b>
AltText (text to substitute for image) .....	346
<b>WebHtmlImageClass Methods</b> .....	<b>347</b>
CreateCellContents (generate HTML image control) .....	347
SetChildDefaults (set image/region relationship) .....	348
SetDescription (set text to substitute for image) .....	348

<b>WebJavaImageClass Properties</b>	<b>349</b>
Filename (image file filename) .....	349
<b>WebJavaImageClass Methods</b>	<b>350</b>
CreateHtml (write HTML for control and its attributes) .....	350
CreateJslData (update Web page control) .....	352
CreateParams (write all image parameters) .....	353
GetAppletType (return applet type) .....	354
<b>WebListClass Methods</b>	<b>355</b>
GetBackgroundColor (return background color) .....	355
<b>WebHtmlListClass Methods</b>	<b>356</b>
CreateCellContents (generate HTML list box) .....	356
CreateJslData (update Web page control) .....	357
ResetControl (update server control) .....	358
<b>WebJavaListClass Properties</b>	<b>359</b>
AutoSpotLink (hypertext links) .....	359
EventActionQ (browser actions for listbox events) .....	359
Format (Web list formatting information) .....	360
FromQ (LIST data source) .....	360
QueueActionQ (Server LIST queue changes) .....	361
Started (Java list applet started) .....	361
<b>WebJavaListClass Methods</b>	<b>362</b>
CreateHtml (write HTML for LIST control) .....	362
CreateJslData (update Web page control) .....	364
CreateParams (write all list parameters) .....	365
GetAppletType (return applet type) .....	366
GetEventAction (return browser action) .....	367
Init (initialize the WebJavaListClass object) .....	368
Kill (shut down the WebJavaListClass object) .....	369
ResetControl (update server control) .....	370
ResetFromQueue (record changes to Server LIST queue) .....	371
SetAutoSpotLink (set live hypertext links) .....	372
SetDirty (force refresh of Web page list box) .....	372
SetEventActin (associate browser action with control event) .....	373
SetQueue (set the data source queue) .....	375
UpdateState (force refresh of Web page list box) .....	376
<b>WebStringClass Methods</b>	<b>377</b>
SetBreakable (allow word wrap) .....	377

<b>WebHtmlStringClass Methods</b>	<b>378</b>
CreateCellContents (generate HTML text string) .....	378
GetCellAttributes (return control attributes) .....	379
<b>WebJavaStringClass Properties</b>	<b>380</b>
AutoSpotLink (hypertext links) .....	380
LastText (last transmitted value) .....	380
<b>WebJavaStringClass Methods</b>	<b>381</b>
CreateHtml (write HTML for control and its attributes) .....	381
CreateJsldata (update Web page control) .....	382
CreateParams (write all string parameters) .....	383
GetAppletType (return applet type) .....	384
Init (initialize WebJavaStringClass object) .....	385
SetAutoSpotLink (set live hypertext links) .....	386
<b>WebCloseButtonClass Properties</b>	<b>387</b>
Height (button height) .....	387
Width (button width) .....	387
X (button horizontal position) .....	387
Y (button vertical position) .....	387
<b>WebCloseButtonClass Methods</b>	<b>388</b>
CreateHtml (write HTML for control and its attributes) .....	388
CreateJsldata (update Web page control) .....	390
CreateParams (write all string parameters) .....	391
GetAppletType (return applet type) .....	392
GetCloneFeq (return button to mimic) .....	393
GetPosition (get control coordinates) .....	394
GetVisible (return control status flag) .....	395
Init (initialize WebCloseButtonClass object) .....	396
ResetControl (apply Web page button action) .....	397
<b>WebHotlinkClass Methods</b>	<b>398</b>
CreateCellContents (generate HTML hypertext link) .....	398
<b>WebLiteralClass Properties</b>	<b>399</b>
Text (Web page text) .....	399
<b>WebLiteralClass Methods</b>	<b>400</b>
CreateHtml (write HTML for control) .....	400
GetCellAttributes (return control attributes) .....	401

<b>WebNullControlClass Methods</b>	<b>402</b>
CreateHtml (write HTML for control) .....	402
CreateJsldata (update Web page control) .....	402
GetAppletType (return applet type).....	403
GetCellAttributes (return control attributes) .....	404
GetIsChild (return family identity).....	405
GetVisible (return control status flag) .....	406

## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

## WebControlClass Concepts

---

The WebControlClass analyzes a single Clarion control and generates the HTML code to duplicate the Clarion control on a Web page. It manages all the control attributes and behaviors that are common to all the Clarion controls.

## Classes Derived from WebControlClass

---

Each type of Clarion control has unique HTML code requirements. For example, the HTML code generated for a CHECK is different than for an ENTRY. Therefore, each type of Clarion control has a corresponding class derived from the WebControlClass to implement its type-specific behavior. For example, the WebHtmlPromptClass represents a PROMPT control, and the WebHtmlEntryClass represents an ENTRY control, etc. These derived classes are documented in this chapter.

## Relationship to Other Internet Builder Classes

---

All the classes described in this chapter are derived from the WebControlClass; therefore, the information and documentation pertaining to the WebControlClass is directly applicable to these derived classes. Please see *Web Control Class* for more information.

## Internet Connect Template Implementation

---

All the classes described in this chapter are derived from the WebControlClass; therefore, the information and documentation pertaining to the WebControlClass is directly applicable to these derived classes. Please see *Web Control Class* for more information.

## Source Files

---

The source code for the classes in this chapter is installed by default to the \LIBSRC folder. The class declarations are in the following .INC files and their respective method definitions are in the corresponding .CLW files.

```
ICCNTRLS.INC  WebButtonClass
               WebCloseButtonClass
               WebHotlinkClass
               WebHtmlButtonClass
               WebHtmlCheckClass
               WebHtmlEntryClass
               WebHtmlGroupClass
               WebHtmlImageClass
               WebHtmlItemClass
               WebHtmlListClass
               WebHtmlMenuClass
               WebHtmlOptionClass
               WebHtmlPromptClass
               WebHtmlRadioClass
               WebHtmlRegionClass
               WebHtmlSheetClass
               WebHtmlStringClass
               WebHtmlTabClass
               WebHtmlTextClass
               WebImageClass
               WebJavaButtonClass
               WebJavaImageClass
               WebJavaListClass
               WebJavaStringClass
               WebJavaToolButtonClass
               WebListClass
               WebLiteralClass
               WebNullControlClass
               WebStringClass
```

## WebHtmlCheckClass Methods

The WebHtmlCheckClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlCheckClass contains the methods listed below.

### BeforeResetControl (control preprocessing)

#### BeforeResetControl, VIRTUAL

The **BeforeResetControl** method does any control specific processing prior to synchronizing the WINDOW control with its corresponding Web page control.

BeforeResetControl is a VIRTUAL method so that other base class methods can directly call the BeforeResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The BeforeResetControl method clears the CHECK control. This is needed because the Client browser never sends information regarding a cleared check box—no information means the box is not checked—and always sends information regarding a checked box.

The WebWindowClass.TakeRequest method calls BeforeResetControl.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index       SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()          !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.                !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
. . . .
```

See Also: [WebWindowClass.TakeRequest](#)

## CreateCellContents (generate HTML check box)

### CreateCellContents( *html object* ), VIRTUAL

#### CreateCellContents

Generates HTML code to represent a check box.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a check box.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

#### Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)           !write HTML TABLE CELL footer
```

#### See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata PROCEDURE(*JslManagerClass Target)
Index   SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                       !get next control
  IF (SELF.Controls.ThisControl.GetVisible())     !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## GetHasHotkey (control text may contain &)

---

### GetHasHotKey, BYTE, VIRTUAL

The **GetHasHotKey** method returns a value indicating whether the control text may contain the Clarion hot key delimiter (&). A return value of one (1) indicates a possible hot key; a return value of zero (0) indicates no hot key. The `WebControlClass` object uses this method to remove the hot key delimiter from Web page text.

`GetHasHotKey` is a `VIRTUAL` method so that other base class methods can directly call the `GetHasHotKey` virtual method in a derived class. This lets you easily implement your own custom version of this method.

**Implementation:** The `WebControlClass.GetText` method calls the `GetHasHotKey` method to identify controls that may have Clarion hot key delimiters (&).

**Return Data Type:** `BYTE`

**Example:**

```
WebControlClass.GetText    FUNCTION
                             CODE
                             IF (SELF.GetHasHotkey())                !check for hot key
                                 RETURN IC:StripHotkey(CLIP(SELF.Feq{PROP:ScreenText}))!return text sans delimiter
                             END
                             RETURN CLIP(SELF.Feq{PROP:ScreenText})    !return text
```

**See Also:** `WebControlClass.GetText`

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server CHECK control with its corresponding Web page check box.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server CHECK control with its corresponding Web page check box. The ResetControl method takes Web page control information submitted by the Client browser, such as control contents and events, and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.               !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: WebWindowClass.TakeRequest

## WebHtmlEntryClass Methods

The WebHtmlEntryClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlEntryClass contains the methods listed below.

### CreateCellContents (generate HTML entry box)

#### CreateCellContents( *html object* ), VIRTUAL

##### CreateCellContents

Generates HTML code to represent an entry box.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent an entry box.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)           !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata PROCEDURE(*JslManagerClass Target)
Index   SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                       !get next control
  IF (SELF.Controls.ThisControl.GetVisible())     !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server ENTRY control with its corresponding Web page entry box.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server ENTRY control with its corresponding Web page entry box. The **ResetControl** method takes Web page control information submitted by the Client browser, such as control contents and events, and applies it to the corresponding Server control.

**ResetControl** is a VIRTUAL method so that other base class methods can directly call the **ResetControl** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebWindowClass.TakeRequest` method calls the **ResetControl** method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.               !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: `WebWindowClass.TakeRequest`

## *WebHtmlGroupClass Properties*

The `WebHtmlGroupClass` inherits all the properties of the `WebControlClass` from which it is derived.

In addition to (or instead of) the inherited properties, the `WebHtmlGroupClass` contains the properties listed below.

### **BorderWidth (Web page control border width)**

---

<b>BorderWidth</b>	<b>BYTE</b>
--------------------	-------------

The **BorderWidth** property contains the control's border width as it appears on the Web page. A value of zero (0) indicates no border. Larger numbers produce wider borders.

The `BorderWidth` property is only effective for `GROUP` controls with the `BOXED` attribute.

Implementation:

The `Init` method sets the initial value of the `BorderWidth` property equal to the `WebWindowClass.GroupBorderWidth` property. The `SetBorderWidth` method sets subsequent values of the `BorderWidth` property.

See Also:

`Init`, `SetBorderWidth`

## WebHtmlGroupClass Methods

The WebHtmlGroupClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlGroupClass contains the methods listed below.

### CreateHtml (write HTML for GROUP control)

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the GROUP control *and* its child controls.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the GROUP control *and* its child controls.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The CreateHtml method calls the WebWindowClass.CreateChildHtml method to generate HTML representing the GROUP's child controls. This ensures that the GROUP's children are correctly positioned on the Web page and that they inherit properties of the parent GROUP, such as fonts or colors.

The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```
MyWebHtmlGroupClass.CreateHtml  PROCEDURE(*HtmlClass Target)
Border          SIGNED(0)
CODE
IF (SELF.Feq{PROP:boxed}<>0)
  Border = SELF.BorderWidth
END
IF (Border)
  Target.WriteTableHeader(' BORDER=' & Border)
END
SELF.OwnerWindow.CreateChildHtml(Target, SELF.Feq)
IF (Border)
  Target.WriteTableFooter
END
```

See Also:

LayoutHtmlClass.CreateHtml

## GetHasHotkey (control text may contain &)

---

### GetHasHotKey, BYTE, VIRTUAL

The **GetHasHotKey** method returns a value indicating whether the control text may contain the Clarion hot key delimiter (&). A return value of one (1) indicates a possible hot key; a return value of zero (0) indicates no hot key. The `WebControlClass` object uses this method to remove the hot key delimiter from Web page text.

`GetHasHotKey` is a `VIRTUAL` method so that other base class methods can directly call the `GetHasHotKey` virtual method in a derived class. This lets you easily implement your own custom version of this method.

**Implementation:** The `WebControlClass.GetText` method calls the `GetHasHotKey` method to identify controls that may have Clarion hot key delimiters (&).

**Return Data Type:** `BYTE`

**Example:**

```
WebControlClass.GetText    FUNCTION
                            CODE
                            IF (SELF.GetHasHotkey())                !check for hot key
                                RETURN IC:StripHotkey(CLIP(SELF.Feq{PROP:ScreenText}))!return text sans delimiter
                            END
                            RETURN CLIP(SELF.Feq{PROP:ScreenText})    !return text
```

**See Also:** `WebControlClass.GetTex`

## Init (initialize WebHtmlGroupClass object)

---

**Init**( *control number*, *owner window* ), **VIRTUAL**

<b>Init</b>	Initializes the WebHtmlGroupClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate).
<i>owner window</i>	The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebHtmlGroupClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The Init method calls the WebControlClass.Init method, and sets the initial value of the BorderWidth property equal to the WebWindowClass.GroupBorderWidth property.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)

CODE
ASSERT(~NewControl &= NULL)
NewControl.Init(Feq, SELF)
SELF.AddControl(NewControl)
```

See Also:

BorderWidth, WebWindowClass.AddControl, WebControlClass.Init

## SetBorderWidth (set Web page control border width)

---

### SetBorderWidth( *width* ), VIRTUAL

**SetBorderWidth** Sets the border width of the control.

*width* An integer constant, variable, EQUATE or expression assigned to the BorderWidth property.

The **SetBorderWidth** method sets the border width of the control as it appears on the Web page. The SetBorderWidth method is only effective for GROUP controls with the BOXED attribute.

SetBorderWidth is a VIRTUAL method so that other base class methods can directly call the SetBorderWidth virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The SetBorderWidth method sets the BorderWidth property.

See Also: BorderWidth

## WebHtmlItemClass Methods

The WebHtmlItemClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlItemClass contains the methods listed below.

### CreateCellContents (generate HTML menu item)

#### CreateCellContents( *html object* ), VIRTUAL

##### CreateCellContents

Generates HTML code to represent a menu item.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a menu item.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)          !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the menu item should appear on the Web page. A return value of one (1) indicates the item should appear on the Web page; a return value of zero (0) indicates the item should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: A menu item may be omitted from the Web page because it is disabled or hidden, or its parent is disabled or hidden.

Return Data Type: **BYTE**

Example:

```
WebControlListClass.AddControlsToLayout PROCEDURE |  
    (*WebControlQueue Source, *LayoutHtmlClass Layout)
```

```
CurIndex          SIGNED,AUTO
```

```
CODE  
LOOP CurIndex = 1 TO RECORDS(Source)  
    GET(Source, CurIndex)  
    IF (Source.ThisControl.GetVisible())  
        Layout.Insert(Source.ThisControl)  
    END  
END  
END
```

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server ITEM control with its corresponding Web page menu item.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server ITEM control with its corresponding Web page menu item. The ResetControl method takes Web page control information submitted by the Client browser, such as an accepted event, and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index       SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()          !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.                !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: WebWindowClass.TakeRequest

## WebHtmlMenuClass Methods

The WebHtmlMenuClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlMenuClass contains the methods listed below.

### CreateCellContents (generate HTML menu)

#### CreateCellContents( *html object* ), VIRTUAL

##### CreateCellContents

Generates HTML code to represent a menu.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a menu.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)          !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## CreateHtml (write HTML for MENU control)

---

### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the MENU control *and* its child controls.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the MENU control *and* its child controls.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The CreateHtml method calls the WebWindowClass.CreateChildHtml method to generate HTML representing the MENU's child controls. This ensures that the MENU's children are correctly positioned on the Web page and that they inherit properties of the parent MENU, such as fonts or colors.

The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```
MyWebHtmlMenuClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE  
PARENT.CreateHtml(Target)  
SELF.OwnerWindow.CreateChildHtml(Target, SELF.Feq)
```

See Also:

LayoutHtmlClass.CreateHtml

## GetCellAttributes (return control attributes)

**GetCellAttributes**( *html object* ), **STRING**, **VIRTUAL**

**GetCellAttributes** Returns attributes associated with the Web page menu control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **GetCellAttributes** method returns attributes associated with the Web page menu control.

GetCellAttributes is a VIRTUAL method so that other base class methods can directly call the GetCellAttributes virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetCellAttributes method returns the font typeface, size, color, and style of the HTML TABLE CELL containing the control, plus nowrap and alignment specifications.

The LayoutHtmlClass.CreateHtml method calls the GetCellAttributes method.

Return Data Type: **STRING**

Example:

```
MyLayoutHtmlClass.CreateHtml  PROCEDURE(*HtmlClass Target)

CurCell      &LayoutCellClass,AUTO
Index         SIGNED,AUTO
NumRows      SIGNED,AUTO
Xindex       SIGNED,AUTO
Yindex       SIGNED,AUTO

CODE
NumRows = RECORDS(SELF.Rows)
Target.Write('<<TABLE' & SELF.Style & '>')
LOOP Yindex = 1 TO NumRows
  GET(SELF.Rows, Yindex)
  Target.Write('<<TR>')
  LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)
    GET(SELF.Rows.Columns, Xindex)
    CurCell &= SELF.Rows.Columns.Cell
    Target.Write('<<TD' & CurCell.GetCellAttributes() & '>')
    Target.WriteLine('<</TD>')
  END
  Target.Write('<</TR>')
END
Target.WriteLine('<</TABLE>')
```

See Also: **LayoutHtmlClass.CreateHtml**

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the menu should appear on the Web page. A return value of one (1) indicates the menu should appear on the Web page; a return value of zero (0) indicates the menu should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: A menu may be omitted from the Web page because it has no children, or because it is explicitly omitted from the Web page based on the WebWindowBaseClass.MenubarType property.

Return Data Type: **BYTE**

Example:

```
WebControlListClass.AddControlsToLayout PROCEDURE |  
    (*WebControlQueue Source, *LayoutHtmlClass Layout)
```

```
CurIndex          SIGNED,AUTO
```

```
CODE  
LOOP CurIndex = 1 TO RECORDS(Source)  
    GET(Source, CurIndex)  
    IF (Source.ThisControl.GetVisible())  
        Layout.Insert(Source.ThisControl)  
    END  
END
```

See Also: **WebWindowBaseClass.MenubarType**

## *WebHtmlOptionClass Properties*

The `WebHtmlOptionClass` inherits all the properties of the `WebControlClass` from which it is derived.

In addition to (or instead of) the inherited properties, the `WebHtmlOptionClass` contains the properties listed below.

### **BorderWidth (Web page control border width)**

---

<b>BorderWidth</b>	<b>BYTE</b>
--------------------	-------------

The **BorderWidth** property contains the control's border width as it appears on the Web page. A value of zero (0) indicates no border. Larger numbers produce wider borders.

The `BorderWidth` property is only effective for `OPTION` controls with the `BOXED` attribute.

Implementation:

The `Init` method sets the initial value of the `BorderWidth` property equal to the `WebWindowClass.OptionBorderWidth` property. The `SetBorderWidth` method sets subsequent values of the `BorderWidth` property.

See Also:

`Init`, `SetBorderWidth`

## WebHtmlOptionClass Methods

The WebHtmlOptionClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlOptionClass contains the methods listed below.

### CreateHtml (write HTML for OPTION control)

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the OPTION control *and* its child controls.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the OPTION control *and* its child controls.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The CreateHtml method calls the WebWindowClass.CreateChildHtml method to generate HTML representing the OPTION's child controls. This ensures that the OPTION's children are correctly positioned on the Web page and that they inherit properties of the parent OPTION, such as fonts or colors.

The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```
WebHtmlOptionClass.CreateHtml PROCEDURE(*HtmlClass Target)
Border          SIGNED(0)
CODE
IF (SELF.Feq{PROP:boxed}<>0)
  Border = SELF.BorderWidth
END
IF (Border)
  Target.WriteTableHeader(' BORDER=' & Border)
END
SELF.OwnerWindow.CreateChildHtml(Target, SELF.Feq)
IF (Border)
  Target.WriteTableFooter
END
SELF.UpdateCopyChoice
```

See Also: [LayoutHtmlClass.CreateHtml](#)

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data for the option/radio buttons control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata PROCEDURE(*JslManagerClass Target)
Index   SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                         !get next control
  IF (SELF.Controls.ThisControl.GetVisible())      !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## GetTableAttributes (return HTML STYLE)

---

### GetTableAttributes, STRING, VIRTUAL

The **GetTableAttributes** method returns the HTML <STYLE> string for the HTML <TABLE> representing the OPTION control.

GetTableAttributes is a VIRTUAL method so that other base class methods can directly call the GetTableAttributes virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The <STYLE> returned by GetTableAttributes includes only the <TABLE> <BORDER>.

Return Data Type:     STRING

Example:

```
WebHtmlOptionClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
  CODE
```

```
  SELF.OwnerWindow.CreateChildHtml(Target, SELF.Feq, SELF.GetTableAttributes())
```

```
  SELF.UpdateCopyChoice
```

## Init (initialize WebHtmlOptionClass object)

---

**Init**( *control number*, *owner window* ), **VIRTUAL**

<b>Init</b>	Initializes the WebHtmlOptionClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate).
<i>owner window</i>	The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebHtmlOptionClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The Init method calls the WebControlClass.Init method, and sets the initial value of the BorderWidth property equal to the WebWindowClass.OptionBorderWidth property.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)

CODE
ASSERT(~NewControl &= NULL)
NewControl.Init(Feq, SELF)
SELF.AddControl(NewControl)
```

See Also:

BorderWidth, WebWindowClass.AddControl, WebControlClass.Init

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server OPTION control with the corresponding Web page option/radio buttons.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server OPTION control with its corresponding Web page option/radio buttons. The **ResetControl** method takes Web page control information submitted by the Client browser, such as selected item and events, and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebWindowClass.TakeRequest` method calls the `ResetControl` method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()          !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.                !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: `WebWindowClass.TakeRequest`

## SetBorderWidth (set Web page control border width)

---

### SetBorderWidth( *width* ), VIRTUAL

**SetBorderWidth** Sets the border width of the control.

*width* An integer constant, variable, EQUATE or expression assigned to the BorderWidth property.

The **SetBorderWidth** method sets the border width of the control as it appears on the Web page. The SetBorderWidth method is only effective for OPTION controls with the BOXED attribute.

SetBorderWidth is a VIRTUAL method so that other base class methods can directly call the SetBorderWidth virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The SetBorderWidth method sets the BorderWidth property.

See Also: BorderWidth

## WebHtmlPromptClass Methods

The WebHtmlPromptClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlPromptClass contains the methods listed below.

### CreateCellContents (generate HTML prompt)

#### CreateCellContents( *html object* ), VIRTUAL

##### CreateCellContents

Generates HTML code to represent a prompt.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a prompt.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

A prompt is simply text on the Web page. The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)           !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## GetHasHotkey (control text may contain &)

---

### GetHasHotKey, BYTE, VIRTUAL

The **GetHasHotKey** method returns a value indicating whether the control text may contain the Clarion hot key delimiter (&). A return value of one (1) indicates a possible hot key; a return value of zero (0) indicates no hot key. The WebControlClass object uses this method to remove the hot key delimiter from Web page text.

GetHasHotKey is a VIRTUAL method so that other base class methods can directly call the GetHasHotKey virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebControlClass.GetText method calls the GetHasHotKey method to identify controls that may have Clarion hot key delimiters (&).

Return Data Type: BYTE

Example:

```
WebControlClass.GetText    FUNCTION
    CODE
    IF (SELF.GetHasHotkey())           !check for hot key
        RETURN IC:StripHotkey(CLIP(SELF.Feq{PROP:ScreenText}))!return text sans delimiter
    END
    RETURN CLIP(SELF.Feq{PROP:ScreenText})           !return text
```

See Also: WebControlClass.GetText

## WebHtmlRadioClass Methods

The WebHtmlRadioClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlRadioClass contains the methods listed below.

### CreateCellContents (generate HTML radio button)

**CreateCellContents**( *html object* ), VIRTUAL

#### CreateCellContents

Generates HTML code to represent a radio button.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a radio button.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)           !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## GetHasHotkey (control text may contain &)

---

### GetHasHotKey, BYTE, VIRTUAL

The **GetHasHotKey** method returns a value indicating whether the control text may contain the Clarion hot key delimiter (&). A return value of one (1) indicates a possible hot key; a return value of zero (0) indicates no hot key. The `WebControlClass` object uses this method to remove the hot key delimiter from Web page text.

`GetHasHotKey` is a `VIRTUAL` method so that other base class methods can directly call the `GetHasHotKey` virtual method in a derived class. This lets you easily implement your own custom version of this method.

**Implementation:** The `WebControlClass.GetText` method calls the `GetHasHotKey` method to identify controls that may have Clarion hot key delimiters (&).

**Return Data Type:** `BYTE`

**Example:**

```
WebControlClass.GetText    FUNCTION
                            CODE
                            IF (SELF.GetHasHotkey())                !check for hot key
                                RETURN IC:StripHotkey(CLIP(SELF.Feq{PROP:ScreenText}))!return text sans delimiter
                            END
                            RETURN CLIP(SELF.Feq{PROP:ScreenText})    !return text
```

**See Also:** `WebControlClass.GetText`

## WebHtmlRegionClass Methods

The WebHtmlRegionClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlRegionClass contains the methods listed below.

### CreateHtmlExtra (write HTML for related control)

#### CreateHtmlExtra( *html object* ), VIRTUAL

**CreateHtmlExtra** Writes HTML code for a related control.

*html object*            The label of the HtmlClass object that writes the HTML.

The **CreateHtmlExtra** method writes HTML code to complete the representation of a related control. For example, a WebHtmlImageClass object that is the parent of one or more WebHtmlRegionClass objects may call the WebHtmlRegionClass.CreateHtmlExtra method to complete the HTML representation of the IMAGE control.

CreateHtmlExtra is a VIRTUAL method so that other base class methods can directly call the CreateHtmlExtra virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
MyWebHtmlImageClass.CreateCellContents PROCEDURE(*HtmlClass Target)
Children            WebControlQueue
Filename            CSTRING(FILE:MaxFileName)
Id                  UNSIGNED,AUTO
ImagePos            LIKE(Rect)
CODE
Filename = SELF.OwnerWindow.Files.GetAlias(SELF.Feq{PROP:tempimage})
Id = IC:Feq2Id(SELF.Feq)
IF (Filename)
  SELF.OwnerWindow.GetChildren(Children, SELF.Feq)
  IF RECORDS(Children)
    Target.WriteLine('<<MAP NAME="MAP' & Id & '">')
    SELF.OwnerWindow.CreateHtmlExtra(Children, Target)
    Target.WriteLine('<</MAP>')
  END
  Target.Write('<<IMG SRC="" & Filename & ""')
  IF (RECORDS(Children))
    Target.Write(' USEMAP="#MAP' & Id & ""')
  END
  GetPosition(SELF.Feq,,ImagePos.width,ImagePos.height)
  Target.Write(' WIDTH=' & Target.GetPixelsX(ImagePos.width))
  Target.Write(' HEIGHT=' & Target.GetPixelsY(ImagePos.height))
  Target.WriteLine('>')
END
```

See Also:

WebHtmlImageClass.CreateCellContents

## SetParentDefaults (confirm parent)

**SetParentDefaults**( *potential parent*, *coordinates* ), **VIRTUAL**

**SetParentDefaults** Confirms this control's boundaries fall entirely within the *potential parent* boundaries.

*potential parent* The label of a WebControlClass object.

*coordinates* The label of a structure containing the coordinates of the *potential parent*.

The **SetParentDefaults** method confirms this REGION control's boundaries fall entirely within the *potential parent* boundaries, typically an IMAGE. If they do, it sets the ParentFeq property to reflect the parent/child relationship.

The WebControlClass uses this method to establish the correct relationships between IMAGE and REGION controls and SHEET and LIST controls in template generated Browse procedures.

For Web page purposes, a parent/child relationship between controls is defined by the controls' positional coordinates. A control whose boundaries fall entirely within the boundaries of another control is the child of the surrounding control (if there is no intervening surrounding control). This position based relationship allows for correct positioning, alignment and processing of Web page controls.

SetParentDefaults is a VIRTUAL method so that other base class methods can directly call the SetParentDefaults virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The *coordinates* parameter names a GROUP with the same structure as the Rect GROUP declared in ICWINDOW.INC as follows:

```
Rect          GROUP,TYPE
x             SIGNED
y             SIGNED
width        SIGNED
height       SIGNED
END
```

Example:

```
WebControlListClass.SetParentDefaults PROCEDURE|
  (*WebControlQueue Source, *WebControlClass Other, *Rect ParentPos)

CurIndex          SIGNED,AUTO

CODE
LOOP CurIndex = 1 TO RECORDS(Source)
  GET(Source, CurIndex)
  Source.ThisControl.SetParentDefaults(Other, ParentPos)
END
```

See Also:

WebWindowClass.SetParentDefaults

## ***WebHtmlSheetClass Properties***

The `WebHtmlSheetClass` inherits all the properties of the `WebControlClass` from which it is derived.

In addition to (or instead of) the inherited properties, the `WebHtmlSheetClass` contains the properties listed below.

### **BorderWidth (Web page control border width)**

---

<b>BorderWidth</b>	<b>BYTE</b>
--------------------	-------------

The **BorderWidth** property contains the control's border width as it appears on the Web page. A value of zero (0) indicates no border. Larger numbers produce wider borders.

Implementation:

The `Init` method sets the initial value of the `BorderWidth` property equal to the `WebWindowClass.SheetBorderWidth` property. The `SetBorderWidth` method sets subsequent values of the `BorderWidth` property.

See Also:

`Init`, `SetBorderWidth`

## WebHtmlSheetClass Methods

The WebHtmlSheetClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlSheetClass contains the methods listed below.

### CreateHtml (write HTML for SHEETcontrol)

---

**CreateHtml**( *html object* ), VIRTUAL

**CreateHtml**      Writes the HTML code representing the SHEET control *and* its child controls.

*html object*      The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the SHEET control *and* its child controls.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The CreateHtml method calls the CreateTabControl method to generate HTML representing the SHEET's child controls. This ensures that the SHEET's children are correctly positioned on the Web page and that they inherit properties of the parent SHEET, such as fonts or colors.

The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```

MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem      &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)           !for all controls
  GET(SELF.Rows.Columns, Xindex)
  CurCell &= SELF.Rows.Columns.Cell
  NumItems = RECORDS(CurCell.Contents)
  Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
  LOOP Index = 1 TO NumItems                             !for all CELL items
    CurItem &= CurCell.GetItem(Index)                   !set WebControl object
    Style = CurItem.GetCellAttributes(Target)           !get control attributes
    IF (Style)
      Target.Write('<<P ' & Style & '>')             !begin HTML STYLE
    END
    CurItem.CreateHtml(Target)                           !write HTML control
    IF (Style)
      Target.Write('<</P>')                             !end HTML STYLE
    END
  END
  Target.WriteLine('<</TD>')                             !end HTML CELL
END
!procedure code

```

See Also: [LayoutHtmlClass.CreateHtml](#), [CreateTabControl](#)

## CreateTabControl (write HTML for SHEET TABs)

**CreateTabControl**( *tabs*, *html object*, *position* , *selected tab* )

### CreateTabControl

Writes the HTML for the SHEET control's TABs.

<i>tabs</i>	The label of the structure containing information about the SHEET control's TAB controls.
<i>html object</i>	The label of the HtmlClass object that writes the HTML code.
<i>position</i>	An integer constant, variable, EQUATE, or expression indicating the tabs' position relative to the SHEET control. Valid positions are above, left, right, and below.
<i>selected tab</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate) of the selected tab.

The **CreateTabControl** method writes the HTML for the SHEET control's TAB controls.

Implementation:

The CreateTabControl method writes JavaScript to define the SHEET's TABs as a group of Java applets within a Java container applet.

The *tabs* parameter names a QUEUE with the same structure as the WebControlQueue declared in ICWINDOW.INC as follows:

```
WebControlQueue    QUEUE,TYPE
Freq               SIGNED
ThisControl        &WebControlClass
                  END
```

EQUATEs for the *position* parameter are declared in PROPERTY.CLW:

```
PROP:left         EQUATE(7C08H) ! 0 = off, else on
PROP:right        EQUATE(7C0CH) ! 0 = off, else on
PROP:above        EQUATE(7C0AH) ! 0 = off, else on
PROP:below        EQUATE(7C06H) ! 0 = off, else on
```

Example:

```
MyWebHtmlSheetClass.CreateHtml PROCEDURE(*HtmlClass Target)
Alignment          SIGNED(PROP:above)
Children           WebControlQueue
CurTabFreq        SIGNED,AUTO
TabControls        WebControlQueue
CODE
SELF.OwnerWindow.GetChildren(TabControls, SELF.Feq, CREATE:Tab)
CurTabFreq = SELF.Feq{PROP:choicefreq}
Target.WriteLine('<<TABLE' & SELF.GetTableAttributes() & '>><<TR>')
Target.WriteLine('<<TD>')
SELF.CreateTabControl(TabControls, Target, Alignment, CurTabFreq)
Target.WriteLine('<</TD><</TR><<TR>')
!procedure code
```

## GetIsChild (return family identity)

**GetIsChild( *parent control* [, *control type*] ), SIGNED, VIRTUAL**

<b>GetIsChild</b>	Returns a value indicating whether this WebControlClass object is a visible child control of the specified <i>parent control</i> and <i>control type</i> .
<i>parent control</i>	An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.
<i>control type</i>	An integer constant, variable, EQUATE, or expression containing the type of control sought. If omitted, any control type is valid.

The **GetIsChild** method returns a value indicating whether this WebControlClass object is a visible child control of the specified *parent control* and *control type*. If this WebControlClass object meets the specified criteria it returns its own control number (Feq property), otherwise it returns zero (0) to indicate it does not meet the specified criteria.

GetIsChild is a VIRTUAL method so that other base class methods can directly call the GetIsChild virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.GetFirstChild method calls the GetIsChild method to find first child controls.

EQUATES for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE:.

Return Data Type: SIGNED

Example:

```
WebWindowClass.GetFirstChild      PROCEDURE(SIGNED ParentFeq, SIGNED Type=0)
Index          SIGNED,AUTO
FirstChild     SIGNED(0)
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.ParentFeq = ParentFeq)
    FirstChild = SELF.Controls.ThisControl.GetIsChild(ParentFeq, Type)
    IF (FirstChild)
      BREAK
    END
  END
END
END
RETURN FirstChild
```

See Also: Feq, WebWindowClass.GetFirstChild

## GetTableAttributes (return HTML STYLE)

---

### GetTableAttributes, STRING, VIRTUAL

The **GetTableAttributes** method returns the HTML <STYLE> string for the HTML <TABLE> representing the SHEET control.

GetTableAttributes is a VIRTUAL method so that other base class methods can directly call the GetTableAttributes virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The <STYLE> returned by GetTableAttributes includes only the <TABLE> <BORDER>.

Return Data Type: STRING

Example:

```
MyWebHtmlSheetClass.CreateHtml PROCEDURE(*HtmlClass Target)
Alignment          SIGNED(PROP:above)
Children           WebControlQueue
CurTabFeq         SIGNED,AUTO
TabControls        WebControlQueue
CODE
SELF.OwnerWindow.GetChildren(TabControls, SELF.Feq, CREATE:Tab)
CurTabFeq = SELF.Feq{PROP:choicefeq}
Target.WriteLine('<<TABLE' & SELF.GetTableAttributes() & '>><<TR>')
Target.WriteLine('<<TD>')
SELF.CreateTabControl(TabControls, Target, Alignment, CurTabFeq)
Target.WriteLine('<</TD><</TR><<TR>')
!procedure code
```

## Init (initialize WebHtmlSheetClass object)

---

**Init**( *control number*, *owner window object* ), **VIRTUAL**

**Init** Initializes the WebHtmlSheetClass object.

*control number* An integer constant, variable, EQUATE, or expression containing the control number (field equate).

*owner window object* The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebHtmlSheetClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The Init method calls the WebControlClass.Init method, and sets the initial value of the BorderWidth property equal to the WebWindowClass.SheetBorderWidth property.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)
```

```
CODE  
ASSERT(~NewControl &= NULL)  
NewControl.Init(Feq, SELF)  
SELF.AddControl(NewControl)
```

See Also: [BorderWidth](#), [WebWindowClass.AddControl](#), [WebControlClass.Init](#)

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server SHEET control with the corresponding Web page controls.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server SHEET control with its corresponding Web page controls. The ResetControl method takes Web page control information submitted by the Client browser, such as the currently selected tab, and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index       SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()          !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.                !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: WebWindowClass.TakeRequest

## SetBorderWidth (set Web page control border width)

---

### SetBorderWidth( *width* ), VIRTUAL

**SetBorderWidth** Sets the border width of the control.

*width* An integer constant, variable, EQUATE or expression assigned to the BorderWidth property.

The **SetBorderWidth** method sets the border width of the control as it appears on the Web page.

SetBorderWidth is a VIRTUAL method so that other base class methods can directly call the SetBorderWidth virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The SetBorderWidth method sets the BorderWidth property.

See Also: BorderWidth

## SetChildDefaults (set nested child controls)

---

### SetChildDefaults, VIRTUAL

The **SetChildDefaults** method sets the children of this control, considering only controls that have the same parent. This has the effect of properly relating layered (stacked) controls, nested controls, or both. For example a REGION control on top of an IMAGE control (layered to generated events for the IMAGE), or a SHEET control and its TAB controls on top of a LIST control (layered and nested to give the appearance of a LIST for each TAB), or a BUTTON control within an OPTION control within a GROUP control (simple nesting to give proper visual organization of related controls).

SetChildDefaults is a VIRTUAL method so that other base class methods can directly call the SetChildDefaults virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: SetChildDefaults calls the WebControlClass.DoSetChildren method. The WebWindowClass.SetChildDefaults method calls the SetChildDefaults method.

Example:

```
WebWindowClass.SetChildDefaults  PROCEDURE
Index                               SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  SELF.Controls.ThisControl.SetChildDefaults
END
```

See Also: WebControlClass.DoSetChildren, WebWindowClass.SetChildDefaults

## ***WebHtmlTabClass Properties***

The WebHtmlTabClass inherits all the properties of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlTabClass contains the properties listed below.

### **IsEnabled (control enabled flag)**

---

<b>IsEnabled</b>	<b>BYTE</b>
------------------	-------------

The **IsEnabled** property indicates whether the control is enabled or disabled. A value of one (1) indicates the control is enabled; a zero (0) indicates the control is disabled. The WebHtmlTabClass uses this property to enable or disable the Web page control corresponding to the Server TAB control.

Implementation:

The CreateHtml method sets the value of the IsEnabled property as needed.

See Also:

CreateHtml

## WebHtmlTabClass Methods

The WebHtmlTabClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlTabClass contains the methods listed below.

### CreateHtml (write HTML for control and its attributes)

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml**      Writes the HTML code representing the TAB control and its attributes.

*html object*      The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the TAB control and its attributes.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:      The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```
MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem            &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)            !for all controls
GET(SELF.Rows.Columns, Xindex)
CurCell &= SELF.Rows.Columns.Cell
NumItems = RECORDS(CurCell.Contents)
Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
LOOP Index = 1 TO NumItems            !for all CELL items
CurItem &= CurCell.GetItem(Index)            !set WebControl object
Style = CurItem.GetCellAttributes(Target)            !get control attributes
IF (Style)
Target.Write('<<P ' & Style & '>')            !begin HTML STYLE
END
CurItem.CreateHtml(Target)            !write HTML control
IF (Style)
Target.Write('<</P>')            !end HTML STYLE
END
END
Target.WriteLine('<</TD>')            !end HTML CELL
END
```

See Also:            [LayoutHtmlClass.CreateHtml](#)

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreateJsldata method enables or disables the Web page control that corresponds to the TAB control.

The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata  PROCEDURE(*JslManagerClass Target)

Index  SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                         !get next control
  IF (SELF.Controls.ThisControl.GetVisible())      !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## CreateParams (write all tab parameters)

---

### CreateParams( *html object* ), VIRTUAL

**CreateParams** Writes all the applet parameters for the TAB control.  
*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateParams** method writes all the applet parameters for the TAB control.

CreateParams is a VIRTUAL method so that other base class methods can directly call the CreateParams virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
WebHtmlTabClass.CreateHtml PROCEDURE(*HtmlClass Target)
CharToDlgX EQUATE(4)
TabExtraWidth EQUATE(8)
CurText ANY
Filename ANY
Height SIGNED,AUTO
Width SIGNED,AUTO
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
CurText = SELF.GetText()
Width = LEN(CurText) * CharToDlgX + TabExtraWidth
Height = 14
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the TAB control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

GetAppletType is a VIRTUAL method so that other base class methods can directly call the GetAppletType virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetAppletType method returns 'ClarionImageButton.'

Return Data Type: STRING

Example:

```
WebHtmlTabClass.CreateHtml PROCEDURE(*HtmlClass Target)
CharToDlgX EQUATE(4)
TabExtraWidth EQUATE(8)
CurText ANY
Filename ANY
Height SIGNED,AUTO
Width SIGNED,AUTO
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
CurText = SELF.GetText()
Width = LEN(CurText) * CharToDlgX + TabExtraWidth
Height = 14
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

## GetHasHotkey (control text may contain &)

---

### GetHasHotKey, BYTE, VIRTUAL

The **GetHasHotKey** method returns a value indicating whether the control text may contain the Clarion hot key delimiter (&). A return value of one (1) indicates a possible hot key; a return value of zero (0) indicates no hot key. The `WebControlClass` object uses this method to remove the hot key delimiter from Web page text.

`GetHasHotKey` is a `VIRTUAL` method so that other base class methods can directly call the `GetHasHotKey` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebControlClass.GetText` method calls the `GetHasHotKey` method to identify controls that may have Clarion hot key delimiters (&).

Return Data Type: **BYTE**

Example:

```
WebControlClass.GetText    FUNCTION
    CODE
    IF (SELF.GetHasHotkey())           !check for hot key
        RETURN IC:StripHotkey(CLIP(SELF.Feq{PROP:ScreenText}))!return text sans delimiter
    END
    RETURN CLIP(SELF.Feq{PROP:ScreenText})           !return text
```

See Also: [WebControlClass.GetText](#)

## GetIsChild (return family identity)

**GetIsChild( *parent control* [, *control type*] ), SIGNED, VIRTUAL**

<b>GetIsChild</b>	Returns a value indicating whether this WebControlClass object is a visible child control of the specified <i>parent control</i> and <i>control type</i> .
<i>parent control</i>	An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.
<i>control type</i>	An integer constant, variable, EQUATE, or expression containing the type of control sought. If omitted, any control type is valid.

The **GetIsChild** method returns a value indicating whether this WebControlClass object is a visible child control of the specified *parent control* and *control type*. If this WebControlClass object meets the specified criteria it returns its own control number (Feq property), otherwise it returns zero (0) to indicate it does not meet the specified criteria.

GetIsChild is a VIRTUAL method so that other base class methods can directly call the GetIsChild virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.GetFirstChild method calls the GetIsChild method to find first child controls.

EQUATES for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE:.

Return Data Type: SIGNED

Example:

```
WebWindowClass.GetFirstChild      PROCEDURE(SIGNED ParentFeq, SIGNED Type=0)
Index          SIGNED,AUTO
FirstChild     SIGNED(0)
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.ParentFeq = ParentFeq)
    FirstChild = SELF.Controls.ThisControl.GetIsChild(ParentFeq, Type)
    IF (FirstChild)
      BREAK
    END
  END
END
END
RETURN FirstChild
```

See Also: Feq, WebWindowClass.GetFirstChild

## GetPosition (get control coordinates)

---

### **GetPosition**( *x*, *y*, *width*, *height* ), VIRTUAL

<b>GetPosition</b>	Returns the control coordinates.
<i>x</i>	A numeric variable to receive the control's horizontal position.
<i>y</i>	A numeric variable to receive the control's vertical position.
<i>width</i>	A numeric variable to receive the control's width.
<i>height</i>	A numeric variable to receive the control's height.

The **GetPosition** method returns the control's SHEET coordinates. The `WebWindowClass` and the `WebControlClass` use this information to help set appropriate position-based parent/child relationships for the controls. The `LayoutHtmlClass` uses this information to help position the control on the Web page.

`GetPosition` is a VIRTUAL method so that other base class methods can directly call the `GetPosition` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
WebControlClass.DoSetChildDefaults PROCEDURE
Children    WebControlQueue
MyRect      GROUP(Rect)
            END
CODE
SELF.GetPosition(MyRect.x, MyRect.y, MyRect.width, MyRect.height)
SELF.OwnerWindow.GetChildren(Children, SELF.ParentFeg)
SELF.OwnerWindow.SetParentDefaults(Children, SELF, MyRect)
```

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the control should appear on the Web page. A return value of one (1) indicates the control should appear on the Web page; a return value of zero (0) indicates the control should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: A TAB control may be omitted from the Web page because it is disabled or hidden.

Return Data Type: BYTE

Example:

```
WebControlListClass.AddControlsToLayout PROCEDURE |
    (*WebControlQueue Source, *LayoutHtmlClass Layout)

CurIndex          SIGNED,AUTO

CODE
LOOP CurIndex = 1 TO RECORDS(Source)
    GET(Source, CurIndex)
    IF (Source.ThisControl.GetVisible())
        Layout.Insert(Source.ThisControl)
    END
END
```

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server TAB control with its corresponding Web page control.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server TAB control with its corresponding Web page control. The **ResetControl** method takes Web page control information submitted by the Client browser, such as enabled state or events, and applies it to the corresponding Server control.

**ResetControl** is a VIRTUAL method so that other base class methods can directly call the **ResetControl** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebWindowClass.TakeRequest` method calls the **ResetControl** method.

Example:

```
MyWebWindowClass.TakeRequest PROCEDURE
NextSubmit &SubmitItemClass
CurFeq    SIGNED,AUTO
Index      SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                                 !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()           !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.                 !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                         !set corresponding control
  IF (NextSubmit.Event)                              !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: `WebWindowClass.TakeRequest`

## SetParentDefaults (confirm parent)

**SetParentDefaults**( *potential parent*, *coordinates* ), **VIRTUAL**

**SetParentDefaults** Confirms this control's boundaries fall entirely within the *potential parent* boundaries.

*potential parent* The label of a WebControlClass object.

*coordinates* The label of a structure containing the coordinates of the *potential parent*.

The **SetParentDefaults** method confirms this control's boundaries fall entirely within the *potential parent* boundaries, and if so, sets the ParentFeq property to reflect the parent/child relationship.

The WebControlClass uses this method to establish the correct parent/child relationships between SHEET controls and TAB controls.

SetParentDefaults is a VIRTUAL method so that other base class methods can directly call the SetParentDefaults virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebHtmlTabClass.SetParentDefaults method does nothing, because, for Web page purposes, the TAB control's children belong to the SHEET control that owns the TAB.

The *coordinates* parameter names a GROUP with the same structure as the Rect GROUP declared in ICWINDOW.INC as follows:

```
Rect          GROUP,TYPE
x             SIGNED
y             SIGNED
width        SIGNED
height       SIGNED
END
```

Example:

```
WebControlListClass.SetParentDefaults PROCEDURE|
  (*WebControlQueue Source, *WebControlClass Other, *Rect ParentPos)

CurIndex          SIGNED,AUTO

CODE
LOOP CurIndex = 1 TO RECORDS(Source)
  GET(Source, CurIndex)
  Source.ThisControl.SetParentDefaults(Other, ParentPos)
END
```

See Also:

WebWindowClass.SetParentDefaults

## WebHtmlTextClass Methods

The WebHtmlTextClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlTextClass contains the methods listed below.

### CreateCellContents (generate HTML text box)

#### CreateCellContents( *html object* ), VIRTUAL

##### CreateCellContents

Generates HTML code to represent a text box.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a text box.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)          !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata PROCEDURE(*JslManagerClass Target)
Index   SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                         !get next control
  IF (SELF.Controls.ThisControl.GetVisible())      !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server CHECK control with its corresponding Web page check box.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server CHECK control with its corresponding Web page check box. The ResetControl method takes Web page control information submitted by the Client browser, such as control contents and events, and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.              !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: WebWindowClass.TakeRequest

## WebButtonClass Methods

The WebHtmlButtonClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlButtonClass contains the methods listed below.

### BeforeResetControl (control preprocessing)

#### BeforeResetControl, VIRTUAL

The **BeforeResetControl** method does any control specific processing prior to synchronizing the WINDOW control with its corresponding Web page control.

BeforeResetControl is a VIRTUAL method so that other base class methods can directly call the BeforeResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The BeforeResetControl method identifies the default button for the window.

The WebWindowClass.TakeRequest method calls BeforeResetControl.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
  IF (SELF.Server.GetRequestedWholePage())
    LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
      GET(SELF.Controls, Index)
      SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
    END
  END
  LOOP                                               !synchronize all controls
    NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
    IF (NextSubmit &= NULL) THEN BREAK.               !stop when finished
    CurFeq = NextSubmit.Feq
    IF (SELF.GetControl(CurFeq))                       !set corresponding control
      IF (NextSubmit.Event)                             !confirm event exists
        IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
          SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
        . . . .
```

See Also:

WebWindowClass.TakeRequest, WebWindowClass.DefaultButton

## GetHasHotkey (control text may contain &)

---

### GetHasHotKey, BYTE, VIRTUAL

The **GetHasHotKey** method returns a value indicating whether the control text may contain the Clarion hot key delimiter (&). A return value of one (1) indicates a possible hot key; a return value of zero (0) indicates no hot key. The `WebControlClass` object uses this method to remove the hot key delimiter from Web page text.

`GetHasHotKey` is a `VIRTUAL` method so that other base class methods can directly call the `GetHasHotKey` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebControlClass.GetText` method calls the `GetHasHotKey` method to identify controls that may have Clarion hot key delimiters (&).

Return Data Type: **BYTE**

Example:

```
WebControlClass.GetText    FUNCTION
    CODE
    IF (SELF.GetHasHotkey())           !check for hot key
        RETURN IC:StripHotkey(CLIP(SELF.Feq{PROP:ScreenText}))!return text sans delimiter
    END
    RETURN CLIP(SELF.Feq{PROP:ScreenText})           !return text
```

See Also: `WebControlClass.GetText`

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the button should appear on the Web page. A return value of one (1) indicates the button should appear on the Web page; a return value of zero (0) indicates the button should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

**Implementation:** The GetVisible method calls the WebControlClass.GetVisible method for buttons implementing standard Windows behavior ({PROP:Std}); otherwise GetVisible returns False.

**Return Data Type:** BYTE

**Example:**

```
WebControlListClass.AddControlsToLayout PROCEDURE |  
    (*WebControlQueue Source, *LayoutHtmlClass Layout)
```

```
CurIndex          SIGNED,AUTO
```

```
CODE  
LOOP CurIndex = 1 TO RECORDS(Source)  
    GET(Source, CurIndex)  
    IF (Source.ThisControl.GetVisible())  
        Layout.Insert(Source.ThisControl)  
    END  
END
```

**See Also:** WebControlClass.GetVisible

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server BUTTON control with its corresponding Web page button.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server BUTTON control with its corresponding Web page button.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The ResetControl method sets the WebWindowClass.DefaultButtonNeeded property.

The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest PROCEDURE
NextSubmit &SubmitItemClass
CurFeq    SIGNED,AUTO
Index     SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()          !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.                !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also:

WebWindowClass.TakeRequest, WebWindowClass.DefaultButtonNeeded

## WebHtmlButtonClass Methods

The WebHtmlButtonClass inherits all the methods of the WebButtonClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlButtonClass contains the methods listed below.

### CreateCellContents (generate HTML button)

#### CreateCellContents( *html object* ), VIRTUAL

##### CreateCellContents

Generates HTML code to represent a button.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a button.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)        !write HTML representing the control
SELF.CreateCellFooter(Target)          !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## Init (initialize WebButtonClass object)

---

**Init**( *control number*, *owner window* ), **VIRTUAL**

<b>Init</b>	Initializes the WebButtonClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate).
<i>owner window</i>	The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebButtonClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The Init method calls the WebControlClass.Init method, then sets PROP:Std to STD:Help for global toolbar help buttons.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)
```

```
CODE  
ASSERT(~NewControl &= NULL)  
NewControl.Init(Feq, SELF)  
SELF.AddControl(NewControl)
```

See Also: [WebControlClass.Init](#)

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server BUTTON control with its corresponding Web page button.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server BUTTON control with its corresponding Web page button. The ResetControl method takes Web page control information submitted by the Client browser, such as control events (EVENT:Accepted), and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The ResetControl method calls the WebButtonClass.ResetControl method before POSTing appropriate events.

The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
  IF (SELF.Server.GetRequestedWholePage())
    LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
      GET(SELF.Controls, Index)
      SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
    END
  END
  LOOP                                               !synchronize all controls
    NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
    IF (NextSubmit &= NULL) THEN BREAK.               !stop when finished
    CurFeq = NextSubmit.Feq
    IF (SELF.GetControl(CurFeq))                       !set corresponding control
      IF (NextSubmit.Event)                             !confirm event exists
        IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
          SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
        . . . . .
```

See Also: WebWindowClass.TakeRequest, WebButtonClass.ResetControl

## *WebJavaButtonClass Properties*

The `WebJavaButtonClass` inherits all the properties of the `WebButtonClass` from which it is derived.

In addition to (or instead of) the inherited properties, the `WebJavaButtonClass` contains the properties listed below.

### **IsEnabled (control enabled flag)**

---

<b>IsEnabled</b>	<b>BYTE</b>
------------------	-------------

The **IsEnabled** property indicates whether the control is enabled or disabled. A value of one (1) indicates the control is enabled; a zero (0) indicates the control is disabled. The `WebJavaButtonClass` uses this property to enable or disable the Web page button corresponding to the Server `BUTTON` control.

Implementation:

The `CreateHtml` method sets the value of the `IsEnabled` property as needed.

See Also:

`CreateHtml`

## WebJavaButtonClass Methods

The WebJavaButtonClass inherits all the methods of the WebButtonClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebJavaButtonClass contains the methods listed below.

### CreateHtml (write HTML for control and its attributes)

**CreateHtml**( *html object* [, *width*, *height*] ), **VIRTUAL**

<b>CreateHtml</b>	Writes the HTML code representing the control and its attributes.
<i>html object</i>	The label of the HtmlClass object that writes the HTML code.
<i>width</i>	An integer constant, variable, EQUATE, or expression indicating the width of the button. If omitted, the WebJavaButtonClass object uses the Clarion AT attribute width.
<i>height</i>	An integer constant, variable, EQUATE, or expression indicating the height of the button. If omitted, the WebJavaButtonClass object uses the Clarion AT attribute height.

The **CreateHtml** method writes the HTML code representing the control and its attributes, including any image associated with the button.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

The CreateHtml method sets the button's width, height, text, image, tooltip, alignment, enabled/disabled state, and the browser action to take when the end user presses the button.

The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```

MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem      &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)           !for all controls
  GET(SELF.Rows.Columns, Xindex)
  CurCell &= SELF.Rows.Columns.Cell
  NumItems = RECORDS(CurCell.Contents)
  Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
  LOOP Index = 1 TO NumItems                             !for all CELL items
    CurItem &= CurCell.GetItem(Index)                   !set WebControl object
    Style = CurItem.GetCellAttributes(Target)           !get control attributes
    IF (Style)
      Target.Write('<<P ' & Style & '>')               !begin HTML STYLE
    END
    CurItem.CreateHtml(Target)                          !write HTML control
    IF (Style)
      Target.Write('<</P>')                             !end HTML STYLE
    END
  END
  Target.WriteLine('<</TD>')                             !end HTML CELL
END

```

See Also:                    **LayoutHtmlClass.CreateHtml**

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreateJsldata method enables or disables the Web page button that corresponds to the BUTTON control.

The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata  PROCEDURE(*JslManagerClass Target)

Index  SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                         !get next control
  IF (SELF.Controls.ThisControl.GetVisible())      !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: [WebWindowClass.CreateJsldata](#)

## CreateParams (write all button parameters)

---

### CreateParams( *html object* ), VIRTUAL

**CreateParams** Writes all the applet parameters for the BUTTON control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateParams** method writes all the applet parameters for the BUTTON control.

CreateParams is a VIRTUAL method so that other base class methods can directly call the CreateParams virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
WebJavaButtonClass.CreateHtml PROCEDURE(*HtmlClass Target,SIGNED Width,SIGNED Height)
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the BUTTON control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

GetAppletType is a VIRTUAL method so that other base class methods can directly call the GetAppletType virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetAppletType method returns 'ClarionImageButton.'

Return Data Type: STRING

Example:

```
WebJavaButtonClass.CreateHtml PROCEDURE(*HtmlClass Target,SIGNED Width,SIGNED Height)
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

## GetCanDisable (return disable-ability flag)

---

### GetCanDisable, BYTE, VIRTUAL

The **GetCanDisable** method returns a value indicating whether the Web page control can be disabled. A return value of one (1) indicates the control can be disabled; a return value of zero (0) indicates the control cannot be disabled.

GetCanDisable is a VIRTUAL method so that other base class methods can directly call the GetCanDisable virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebControlClass.GetVisible` method calls the `GetCanDisable` method. The `WebJavaButtonClass.GetCanDisable` method returns True, because Java buttons can be disabled.

Return Data Type: **BYTE**

Example:

```
WebControlClass.GetVisible          FUNCTION
CODE
IF (SELF.DisabledAction = DISABLE:Hide) OR |
  ((SELF.DisabledAction = DISABLE:OptHide) AND NOT SELF.GetCanDisable())
  IF NOT SELF.Feq{PROP:enabled}
    RETURN FALSE
  END
END
RETURN SELF.Feq{PROP:visible}
```

See Also: **WebControlClass.GetVisible, WebControlClass.DisabledAction**

## GetFilename (return image filename)

---

### GetFilename, STRING, VIRTUAL

The **GetFilename** method returns the filename of the file containing the image to display on the button face.

GetFilename is a VIRTUAL method so that other base class methods can directly call the GetFilename virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The WebJavaButtonClass.CreateHtml method calls the GetFilename method.

Return Data Type:     **STRING**

Example:

```
MyWebJavaButtonClass.CreateHtml       PROCEDURE(*HtmlClass Target)

Filename       CSTRING(FILE:MaxFileName)

CODE
Filename = SELF.GetFilename()
Target.WriteAppletHeader(SELF.Feq, 'ClarionImageButton', Width, Height)
Target.WriteAppletFilenameParameter('Picture', Filename)
Target.WriteAppletOptParameter('Label', SELF.GetText())
Target.WriteAppletOptParameter('Hint', SELF.Feq{PROP:tooltip})
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
IF (SELF.GetEventAction(EVENT:Accepted) = Update:Full)
    Target.WriteAppletParameter('Submit', 1)
END
Target.WriteAppletFooter
```

See Also:             **CreateHtml**

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server BUTTON control with its corresponding Web page button.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server BUTTON control with its corresponding Web page button. The ResetControl method takes Web page control information submitted by the Client browser, such as control events (EVENT:Accepted), and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The ResetControl method calls the WebButtonClass.ResetControl method before POSTing appropriate events.

The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
  IF (SELF.Server.GetRequestedWholePage())
    LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
      GET(SELF.Controls, Index)
      SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
    END
  END
  LOOP                                               !synchronize all controls
  NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
  IF (NextSubmit &= NULL) THEN BREAK.                !stop when finished
  CurFeq = NextSubmit.Feq
  IF (SELF.GetControl(CurFeq))                       !set corresponding control
    IF (NextSubmit.Event)                             !confirm event exists
      IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
        SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
      . . . . .
```

See Also: WebWindowClass.TakeRequest, WebButtonClass.ResetControl

## WebJavaToolButtonClass Methods

The WebJavaToolButtonClass inherits all the methods of the WebJavaButtonClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebJavaToolButtonClass contains the methods listed below.

### GetEventAction (return browser action)

---

#### GetEventAction( *event* ), SIGNED, VIRTUAL

**GetEventAction** Returns the browser action for the specified *event*.

*event* An integer constant, variable, EQUATE, or expression indicating the event for which to return the action.

The **GetEventAction** method returns the browser action associated with the specified *event*. The WebControlClass object uses this method to generate appropriate HTML/JavaScript for the control.

The SetEventAction associates browser actions with events.

GetEventAction is a VIRTUAL method so that other base class methods can directly call the GetEventAction virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebJavaButtonClass.CreateHtml method calls the GetEventAction method.

EQUATEs for the return values are declared in ICSTD.EQU as follows:

Update:OnBrowser	EQUATE(0)
Update:Partial	EQUATE(1)
Update:Full	EQUATE(2)

Return Data Type:

SIGNED

Example:

```
MyWebJavaButtonClass.CreateHtml      PROCEDURE(*HtmlClass Target)

Filename      CSTRING(FILE:MaxFileName)

CODE
Filename = SELF.GetFilename()
Target.WriteAppletHeader(SELF.Feq, 'ClarionImageButton', Width, Height)
Target.WriteAppletFilenameParameter('Picture', Filename)
Target.WriteAppletOptParameter('Label', SELF.GetText())
Target.WriteAppletOptParameter('Hint', SELF.Feq{PROP:tooltip})
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
IF (SELF.GetEventAction(EVENT:Accepted) = Update:Full)
    Target.WriteAppletParameter('Submit', 1)
END
Target.WriteAppletFooter
```

See Also: [WebJavaButtonClass.CreateHtml](#)

## ***WebImageClass***

The `WebImageClass` inherits all the properties and methods of the `WebControlClass` from which it is derived. It has no additional methods or properties, but serves as a foundation to its derived classes: `WebHtmlImageClass` and `WebJavaImageClass`.

## ***WebHtmlImageClass Properties***

The `WebHtmlImageClass` inherits all the properties of the `WebImageClass` from which it is derived.

In addition to (or instead of) the inherited properties, the `WebHtmlImageClass` contains the properties listed below.

### **AltText (text to substitute for image)**

---

#### **AltText CSTRING(255)**

The **AltText** property contains the text to substitute for the Web page image while the image is downloading, or if the end user has elected not to download images.

Implementation: The `SetDescription` method sets the value of the `AltText` property.

See Also: `SetDescription`

## WebHtmlImageClass Methods

The WebHtmlImageClass inherits all the methods of the WebImageClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlCheckClass contains the methods listed below.

### CreateCellContents (generate HTML image control)

**CreateCellContents( *html object* ), VIRTUAL**

#### CreateCellContents

Generates HTML code to display an image.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML to display an image.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The CreateCellContents method writes HTML to represent the IMAGE as well as any REGIONS associated with the IMAGE. In Clarion, REGION controls are often used to generate events for IMAGE controls, which do not generate events. The WebWindowClass deduces the association between the IMAGES and REGIONS based on their (overlapping) coordinates.

WebControlClass.CreateHtml calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLES and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml  PROCEDURE(*HtmlClass Target)
    CODE
    SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
    SELF.CreateCellContents(Target)         !write HTML representing the control
    SELF.CreateCellFooter(Target)          !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## SetChildDefaults (set image/region relationship)

---

### SetChildDefaults, VIRTUAL

The **SetChildDefaults** method sets the children of the Web page image control.

For HTML generation purposes, REGION controls used to generate events for an IMAGE control, become the children of the image control.

SetChildDefaults is a VIRTUAL method so that other base class methods can directly call the SetChildDefaults virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The SetChildDefaults method calls the WebWindowClass.GetChildren method to identify candidate REGION controls. Then it calls the WebWindowClass.SetParentDefaults method to check the candidates' coordinates and confirm the actual children.

See Also:

WebWindowClass.GetChildren, WebWindowClass.SetParentDefaults

## SetDescription (set text to substitute for image)

---

### SetDescription( *text* ), VIRTUAL

**SetDescription** Sets the text to substitute for the image.

*text* A string constant, variable, EQUATE or expression containing the text to substitute for the image.

The **SetDescription** method sets the text to substitute for the Web page image while the image is downloading, or if the end user has elected not to download images.

SetDescription is a VIRTUAL method so that other base class methods can directly call the SetDescription virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The SetDescription method sets the value of the Description property.

See Also:

Description

## *WebJavalmageClass Properties*

The `WebJavaImageClass` inherits all the properties of the `WebImageClass` from which it is derived.

In addition to (or instead of) the inherited properties, the `WebJavaImageClass` contains the properties listed below.

### **Filename (image file filename)**

---

<b>Filename</b>	<b>CSTRING(FILE:MaxFileName)</b>
-----------------	----------------------------------

The **Filename** property contains the filename of the file containing the image to display.

Implementation: The `CreateHtml` method and the `CreateJsldata` method each set the value of the `Filename` property as needed.

See Also: `CreateHtml`, `CreateJsldata`

## ***WebJavaImageClass Methods***

The `WebJavaImageClass` inherits all the methods of the `WebImageClass` from which it is derived.

In addition to (or instead of) the inherited methods, the `WebJavaImageClass` contains the methods listed below.

### **CreateHtml (write HTML for control and its attributes)**

---

#### **CreateHtml( *html object* ), VIRTUAL**

**CreateHtml** Writes the HTML code representing the image control and its attributes.

*html object* The label of the `HtmlClass` object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the image control and its attributes.

`CreateHtml` is a VIRTUAL method so that other base class methods can directly call the `CreateHtml` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The `CreateHtml` method implements the image's width, height, and filename.

The `LayoutHtmlClass.CreateHtml` method calls the `WebControlClass.CreateHtml` method.

Example:

```

MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem      &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)           !for all controls
  GET(SELF.Rows.Columns, Xindex)
  CurCell &= SELF.Rows.Columns.Cell
  NumItems = RECORDS(CurCell.Contents)
  Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
  LOOP Index = 1 TO NumItems                             !for all CELL items
    CurItem &= CurCell.GetItem(Index)                   !set WebControl object
    Style = CurItem.GetCellAttributes(Target)           !get control attributes
    IF (Style)
      Target.Write('<<P ' & Style & '>')               !begin HTML STYLE
    END
    CurItem.CreateHtml(Target)                           !write HTML control
    IF (Style)
      Target.Write('<</P>')                             !end HTML STYLE
    END
  END
  Target.WriteLine('<</TD>')                             !end HTML CELL
END

```

See Also:                    **LayoutHtmlClass.CreateHtml**

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata PROCEDURE(*JslManagerClass Target)
Index   SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                       !get next control
  IF (SELF.Controls.ThisControl.GetVisible())     !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## CreateParams (write all image parameters)

### CreateParams( *html object* ), VIRTUAL

**CreateParams** Writes all the applet parameters for the IMAGE control.  
*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateParams** method writes all the applet parameters for the IMAGE control.

CreateParams is a VIRTUAL method so that other base class methods can directly call the CreateParams virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```

WebJavaImageClass.CreateHtml      PROCEDURE(*HtmlClass Target)
X                                  SIGNED,AUTO
Y                                  SIGNED,AUTO
Width                              SIGNED,AUTO
Height                              SIGNED,AUTO
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
SELF.Filename = SELF.OwnerWindow.Files.GetAlias(SELF.Feq{PROP:tempimage})
IC:GetPositionPixels(SELF.Feq, X,Y,Width,Height)
Target.WriteAppletHeaderPixel(SELF.Feq, SELF.GetAppletType(), Width, Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter

```

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the **BUTTON** control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

**GetAppletType** is a **VIRTUAL** method so that other base class methods can directly call the **GetAppletType** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **GetAppletType** method returns 'ClarionImageControl.'

Return Data Type: **STRING**

Example:

```
WebJavaImageClass.CreateHtml      PROCEDURE(*HtmlClass Target)
X                                  SIGNED,AUTO
Y                                  SIGNED,AUTO
Width                              SIGNED,AUTO
Height                             SIGNED,AUTO
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
SELF.Filename = SELF.OwnerWindow.Files.GetAlias(SELF.Feq{PROP:tempimage})
IC:GetPositionPixels(SELF.Feq, X,Y,Width,Height)
Target.WriteAppletHeaderPixel(SELF.Feq, SELF.GetAppletType(), Width, Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

## WebListClass Methods

The WebListClass inherits all the properties and methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebListClass contains the methods listed below.

### GetBackgroundColor (return background color)

**GetBackgroundColor( [default color] ), LONG, VIRTUAL**

#### GetBackgroundColor

Returns the background color of the Web page list.

*default color* An integer variable, constant, EQUATE, or expression containing the color to return if there is no background color. If omitted, *default color* defaults to Color:None.

The **GetBackgroundColor** method returns the background color of the Web page list.

GetBackgroundColor is a VIRTUAL method so that other base class methods can directly call the GetBackgroundColor virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetBackgroundColor method returns the COLOR attribute of the WINDOW control, if there is one. Otherwise, it returns the background color inherited from the Web page. EQUATES for the *default color* parameter are declared in \LIBSRC\EQUATES.CLW.

The CreateColorParameters method calls GetBackgroundColor.

Return Data Type: LONG

Example:

```
MyWebControlClass.CreateColorParameters PROCEDURE(*HtmlClass Target, BYTE AutoSpotLink)
ForeColor          LONG,AUTO
BackColor          LONG,AUTO
CODE
GETFONT(SELF.Feq,,, ForeColor)
BackColor = SELF.GetBackgroundColor()
IF (ForeColor <> 0) AND (ForeColor <> COLOR:None)
    Target.WriteAppletParameter('ForeColor', IC:RGB(ForeColor))
END
IF (BackColor <> COLOR:None)
    Target.WriteAppletParameter('BackColor', IC:RGB(BackColor))
END
```

See Also: CreateColorParameters

## WebHtmlListClass Methods

The WebHtmlListClass inherits all the methods of the WebListClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlListClass contains the methods listed below.

### CreateCellContents (generate HTML list box)

#### CreateCellContents( *html object* ), VIRTUAL

##### CreateCellContents

Generates HTML code to represent a list box.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a list box.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLES and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)           !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreateJsldata method applies a new selection in the list box.

The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata  PROCEDURE(*JslManagerClass Target)
Index  SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                       !get next control
  IF (SELF.Controls.ThisControl.GetVisible())     !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

<b>ResetControl</b>	Synchronizes the Server LIST control with its corresponding Web page list box.
<i>submit item</i>	The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server LIST control with its corresponding Web page list box. The ResetControl method takes Web page control information submitted by the Client browser, such as a new item selection, and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreateJsldata method applies a new selection in the list box.

The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                               !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()          !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.                !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                       !set corresponding control
  IF (NextSubmit.Event)                             !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . . .
```

See Also: WebWindowClass.TakeRequest

## WebJavaListClass Properties

The WebJavaListClass inherits all the properties of the WebListClass from which it is derived.

In addition to (or instead of) the inherited properties, the WebJavaListClass contains the properties listed below.

### AutoSpotLink (hypertext links)

---

#### AutoSpotLink BYTE, PROTECTED

The **AutoSpotLink** property indicates whether list text that appears to be a URL, email address, FTP site, etc., are implemented on the Web page as “live” hypertext links or plain text. A value of one (1) generates live links; a value of zero (0) generates plain text.

This property is PROTECTED, therefore, it can only be referenced by a WebJavaListClass method, or a method in a class derived from WebJavaListClass.

Implementation: The SetAutoSpotLink method sets the value of the AutoSpotLink property.

See Also: SetAutoSpotLink

### EventActionQ (browser actions for listbox events)

---

#### EventActionQ &EventActionQueue

The **EventActionQ** property is a reference to a structure containing standard Clarion event numbers (such as EVENT:ScrollUp) and their corresponding browser actions (such as Update:Partial). The WebJavaListClass object uses this property to generate appropriate JavaScript for the list box.

Implementation: The Init method creates (allocates memory for) the EventActionQ. The SetEventAction method sets the values of the EventActionQ property. The GetEventAction method returns the action for a specified event. The Kill method disposes of (frees memory for) the EventActionQ.

The EventActionQ property is a reference to a QUEUE with the same structure as EventActionQueue declared in ICEVENT.INC as follows:

```

EventActionQueue   QUEUE, TYPE
EventNo            SIGNED
Action             BYTE
                  END
  
```

See Also: Init, GetEventAction, Kill, SetEventAction

## Format (Web list formatting information)

---

### Format ULONG

The **Format** property contains the Web version of the LIST FORMAT attribute. The WebJavaListClass object passes this formatting information to the Java Support Library.

Implementation: The Init method sets the initial value of the Format property.

See Also: Init

## FromQ (LIST data source)

---

### FromQ &QUEUE

The **FromQ** property is a reference to the LIST's FROM attribute. The WebJavaListClass object uses this property to refer to the LIST's data source QUEUE.

Implementation: The SetQueue method sets the value of the FromQ property. The WebJavaListClass object uses the FromQ property to detect any changes to the source QUEUE, and to avoid generating new JSL data when there are no changes.

See Also: SetQueue

## QueueActionQ (Server LIST queue changes)

### QueueActionQ &QueueActionQueue, PROTECTED

The **QueueActionQ** property is a reference to a structure indicating exactly how the data source QUEUE has changed since the Client browser was last updated. The WebJavaListClass object uses this property to generate minimum JSL data for the list box.

This property is PROTECTED, therefore, it can only be referenced by a WebJavaListClass method, or a method in a class derived from WebJavaListClass.

Implementation:

The Init method creates (allocates memory for) the QueueActionQ. The ResetFromQueue method sets the values of the QueueActionQ property. The Kill method disposes of (frees memory for) the QueueActionQ.

The QueueActionQ property is a reference to a QUEUE with the same structure as QueueActionQueue declared in ICWINDOW.INC as follows:

```
QueueActionQueue    QUEUE,TYPE
Action              BYTE
NumItems            LONG
Offset              LONG
                   END
```

See Also:

Init, Kill, ResetFromQueue

## Started (Java list applet started)

### Started BYTE(False), PROTECTED

The **Started** property indicates whether the Java applet representing the LIST has started. A value of one (1) indicates the applet has started; a value of zero (0) indicates the applet has not started. The WebJavaListClass object uses this property to avoid sending JSL data until the applet is ready to receive it.

This property is PROTECTED, therefore, it can only be referenced by a WebJavaListClass method, or a method in a class derived from WebJavaListClass.

Implementation:

The CreateHtml method sets the Started property to False when it generates code to instantiate the applet; the ResetControl method sets the Started property to True when the applet first requests data.

See Also:

CreateHtml, ResetControl

## WebJavaListClass Methods

The WebJavaListClass inherits all the methods of the WebListClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebJavaListClass contains the methods listed below.

### CreateHtml (write HTML for LIST control)

---

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the LIST control and its attributes.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the LIST control and its attributes.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The CreateHtml method writes list attributes including format string, fonts, colors, text justification, grid lines, scrollbars and more.

The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```

MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem      &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)           !for all controls
  GET(SELF.Rows.Columns, Xindex)
  CurCell &= SELF.Rows.Columns.Cell
  NumItems = RECORDS(CurCell.Contents)
  Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
  LOOP Index = 1 TO NumItems                             !for all CELL items
    CurItem &= CurCell.GetItem(Index)                   !set WebControl object
    Style = CurItem.GetCellAttributes(Target)           !get control attributes
    IF (Style)
      Target.Write('<<P ' & Style & '>')               !begin HTML STYLE
    END
    CurItem.CreateHtml(Target)                           !write HTML control
    IF (Style)
      Target.Write('<</P>')                             !end HTML STYLE
    END
  END
  Target.WriteLine('<</TD>')                             !end HTML CELL
END

```

See Also:                    **LayoutHtmlClass.CreateHtml**

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the CreateJsldata method. The CreateJsldata method writes the minimum amount of data required to refresh the browser list box.

Example:

```
WebWindowClass.CreateJsldata  PROCEDURE(*JslManagerClass Target)

Index  SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                       !get next control
  IF (SELF.Controls.ThisControl.GetVisible())     !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## CreateParams (write all list parameters)

---

### CreateParams( *html object* ), VIRTUAL

**CreateParams** Writes all the applet parameters for the LIST control.  
*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateParams** method writes all the applet parameters for the LIST control.

CreateParams is a VIRTUAL method so that other base class methods can directly call the CreateParams virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
WebJavaListClass.CreateHtml PROCEDURE(*HtmlClass Target)
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
IF (SELF.width = 0)
  GETPOSITION(SELF.Feq,,SELF.width,SELF.height)
END
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),SELF.width,SELF.height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
SELF.Started = FALSE
SELF.UpdateState
```

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the LIST control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

GetAppletType is a VIRTUAL method so that other base class methods can directly call the GetAppletType virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetAppletType method returns 'ClarionListBox.'

Return Data Type: STRING

Example:

```
WebJavaListClass.CreateHtml PROCEDURE(*HtmlClass Target)
  CODE
  IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
  IF (SELF.width = 0)
    GETPOSITION(SELF.Feq,,,SELF.width,SELF.height)
  END
  Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),SELF.width,SELF.height)
  SELF.CreateParams(Target)
  Target.WriteAppletFooter
  SELF.Started = FALSE
  SELF.UpdateState
```

## GetEventAction (return browser action)

### GetEventAction( *event* ), SIGNED, VIRTUAL

**GetEventAction** Returns the browser action for the specified *event*.  
*event* An integer constant, variable, EQUATE, or expression indicating the event for which to return the action.

The **GetEventAction** method returns the browser action associated with the specified *event*. The `WebControlClass` object uses this method to generate appropriate HTML/JavaScript for the control.

The `SetEventAction` associates browser actions with events.

`GetEventAction` is a VIRTUAL method so that other base class methods can directly call the `GetEventAction` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The `WebHtmlListClass.CreateCellContents` method calls the `GetEventAction` method.

EQUATES for the return values are declared in `ICSTD.EQU` as follows:

```
Update:OnBrowser    EQUATE(0)
Update:Partial     EQUATE(1)
Update:Full        EQUATE(2)
```

Return Data Type: **SIGNED**

Example:

```
MyWebHtmlListClass.CreateCellContents    PROCEDURE(*HtmlClass Target)

CODE
Target.Write('<<SELECT' & SELF.GetNameAttribute(Target) & ' SIZE=' & Depth)
IF (SELF.OwnerWindow.AllowJava)
    Target.WriteEventHandler(SELF.GetEventAction(EVENT:Accepted), HTML:SelectChanged,
HTML:SelectValue)
END
Target.WriteLine('>')
Target.Write(IC:GetListboxHtml(SELF.Feq))
Target.WriteLine('<</SELECT>')
SELF.UpdateCopyChoice
```

See Also: `WebHtmlListClass.CreateCellContents`

## Init (initialize the WebJavaListClass object)

---

**Init**( *control number*, *owner window* ), **VIRTUAL**

<b>Init</b>	Initializes the WebJavaListClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate).
<i>owner window</i>	The label of the WebWindowClass object that instantiated this WebJavaListClass object.

The **Init** method initializes the WebJavaListClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebWindowClass.AddControl method calls the Init method. The Init method primes the EventActionQ property with appropriate Clarion events and their corresponding browser actions.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)
```

```
CODE  
ASSERT(~NewControl &= NULL)  
NewControl.Init(Feq, SELF)  
SELF.AddControl(NewControl)
```

See Also:

**EventActionQ**

## Kill (shut down the WebJavaListClass object)

---

### Kill, VIRTUAL

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Kill is a VIRTUAL method so that other base class methods can directly call the Kill virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.Kill method calls the Kill method.

Example:

```
MyWebWindowClass.Kill          PROCEDURE
Index          SIGNED,AUTO
CODE
IF (~SELF.Controls &= NULL)
  LOOP Index = 1 TO RECORDS(SELF.Controls)
    GET(SELF.Controls, Index)
    ASSERT(~SELF.Controls.ThisControl &= NULL)
    SELF.Controls.ThisControl.Kill
  END
DISPOSE(SELF.Controls)
END
```

## ResetControl (update server control)

### ResetControl( *submit item* ), VIRTUAL

**ResetControl** Synchronizes the Server LIST control with its corresponding Web page list box.

*submit item* The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method synchronizes the Server LIST control with its corresponding Web page list box. The ResetControl method takes Web page control information submitted by the Client browser, such as control contents and events, and applies it to the corresponding Server control.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index       SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                           !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.             !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                   !set corresponding control
  IF (NextSubmit.Event)                         !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . .
```

See Also: WebWindowClass.TakeRequest

## ResetFromQueue (record changes to Server LIST queue)

**ResetFromQueue**( *change* [,*offset*] [,*number*] ), **VIRTUAL**

**ResetFromQueue** Records changes to the Server LIST control.

<i>change</i>	An integer constant, variable, EQUATE or expression that indicates the type of change made to the LIST's queue. Valid actions include insert, delete, delete all, replace, scroll, scroll down, and scroll up.
<i>offset</i>	An integer constant, variable, EQUATE or expression that indicates the direction of a scroll action, or the relative position of an insert, delete, or replace action. If omitted, <i>offset</i> defaults to zero (0).
<i>number</i>	An integer constant, variable, EQUATE or expression that indicates the number of list items to scroll for a scroll action; otherwise the number of list items affected for an insert, delete, or replace action. If omitted, <i>number</i> defaults to one (1).

The **ResetFromQueue** method records changes to the Server LIST control's data source queue so the same changes can be efficiently applied to the corresponding Web page control.

ResetFromQueue is a VIRTUAL method so that other base class methods can directly call the ResetFromQueue virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

Typically, this method is called during the scroll handling for the LIST. For example, from the BRWX::ScrollOne ROUTINE for a template generated BrowseBox.

EQUATEs for the how parameter are declared in ICWINDOW.INC as follows:

```

ITEMIZE,PRE(ACTION)
Insert      EQUATE
Delete     EQUATE
Replace    EQUATE
DeleteAll  EQUATE
Scroll     EQUATE
ScrollDown EQUATE
ScrollUp   EQUATE
END

```

## SetAutoSpotLink (set live hypertext links)

---

### SetAutoSpotLink( *value* ), VIRTUAL

**SetAutoSpotLink** Sets the automatic generation of live hypertext links.

*value* A Boolean constant, variable, EQUATE or expression assigned to the AutoSpotLink property.

The **SetAutoSpotLink** method enables or disables the automatic generation of live hypertext links for list text that appears to be a URL, email address, FTP site, etc. A *value* of one (1) enables live links; a *value* of zero (0) disables live links (generates plain text).

SetAutoSpotLink is a VIRTUAL method so that other base class methods can directly call the SetAutoSpotLink virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The SetAutoSpotLink method sets the AutoSpotLink property.

Example:

```
IC:CurControl &= WebWindow.AddControl(?Browse:1)
IC:CurControl.SetAutoSpotLink(TRUE)
```

See Also: AutoSpotLink

## SetDirty (force refresh of Web page list box)

---

### SetDirty

The **SetDirty** method forces a single unconditional generation of new Web page list box data. Once the data is generated, the WebJavaListClass object reverts to conditional generation of data only as needed.

Implementation: The SetDirty method is available for your use. The IBC Library does not call it. The CreateJsldata method generates JSL data to refresh the Web page list box if SetDirty was called.

Example:

```
IC:CurControl &= WebWindow.AddControl(?Browse:1)
IC:CurControl.SetDirty
```

See Also: CreateJsldata

## SetEventAction (associate browser action with control event)

---

### SetEventAction( *event*, *action* ), VIRTUAL

<b>SetEventAction</b>	Associates a browser action with a particular control event.
<i>event</i>	An integer constant, variable, EQUATE or expression identifying the Web control event. Most controls only support EVENT:Accepted, which is triggered when the end user presses an OK or Submit button, or the RETURN, ENTER, or TAB key. However, list based controls also support scrolling events, selection events, and others.
<i>action</i>	An integer constant, variable, EQUATE or expression identifying the browser action. Valid actions are Update:OnBrowser (don't contact the Server ), Update:Partial (request Java control data only from the Server), and Update:Full (request the entire page from the Server).

The **SetEventAction** method associates a browser action with a particular Web page control event. That is, the browser action, such as Update:Partial, is an attribute of the control. When the end user creates an event for the Web page control by mouse-clicking or pressing the RETURN, ENTER, or TAB key, the Client browser requests the action associated with the event. The Clarion Application Broker forwards the request to the Server which processes it.

The GetEventAction method returns the action for a specified event.

SetEventAction is a VIRTUAL method so that other base class methods can directly call the SetEventAction virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: EQUATEs for the *event* parameter are declared in EQUATES.CLW. Each event EQUATE is prefixed with EVENT:. Valid events are

```

EVENT:Accepted
EVENT:Initialize
EVENT:NewSelection
EVENT:AlertKey
EVENT:Locate
EVENT:ScrollTop
EVENT:ScrollBottom
EVENT:PageUp
EVENT:PageDown
EVENT:ScrollUp
EVENT:ScrollDown
EVENT:ScrollDrag
EVENT:Expanding
EVENT:Contracting
EVENT:Expanded
EVENT:Contracted

```

EQUATEs for the *action* parameter are declared in ICSTD.EQU as follows:

```

Update:OnBrowser      EQUATE(0)
Update:Partial        EQUATE(1)
Update:Full           EQUATE(2)

```

Example:

```
WebJavaListClass.Init PROCEDURE(SIGNED Feq, *WebWindowBaseClass OwnerWindow)
```

```

CODE
PARENT.Init(Feq, OwnerWindow)
SELF.EventActionQ &= NEW EventActionQueue
SELF.QueueActionQ &= NEW QueueActionQueue
SELF.SendIcons = TRUE

IF (SELF.feq{PROP:imm})
    SELF.SetEventAction(EVENT:ScrollDown,    Update:Partial)
    SELF.SetEventAction(EVENT:ScrollUp,      Update:Partial)
    SELF.SetEventAction(EVENT:ScrollTop,     Update:Partial)
    SELF.SetEventAction(EVENT:ScrollBottom,  Update:Partial)
    SELF.SetEventAction(EVENT:ScrollDrag,    Update:Partial)
    SELF.SetEventAction(EVENT:PageUp,        Update:Partial)
    SELF.SetEventAction(EVENT:PageDown,      Update:Partial)
    SELF.SetEventAction(EVENT:Locate,        Update:Partial)
END

SELF.SetEventAction(EVENT:NewSelection,     Update:Partial)
SELF.SetEventAction(EVENT:Initialize,       Update:Partial)
SELF.SetEventAction(EVENT:AlertKey,         Update:Partial)

```

See Also: **GetEventAction**

## SetQueue (set the data source queue)

---

### SetQueue( *source* ), VIRTUAL

<b>SetQueue</b>	Sets the WebJavaListClass object's data source queue.
<i>source</i>	The label of the data source QUEUE (the LIST's FROM attribute).

The **SetQueue** method sets the WebJavaListClass object's data source queue, so the object can detect changes to the *source*, and generate the minimum JSL data needed to reflect those changes.

SetQueue is a VIRTUAL method so that other base class methods can directly call the SetQueue virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:      The SetQueue method sets the value of the FromQ property. The FromQ property is a reference to the data source for the LIST control.

See Also:              FromQ

## UpdateState (force refresh of Web page list box)

### UpdateState

The **UpdateState** method saves the current state of various LIST attributes, so the WebJavaListClass object can detect changes to the LIST and generate the minimum JSL data needed to reflect those changes.

Implementation: The UpdateState method stores the HSCROLL, VSCROLL, and FORMAT attributes of the LIST.

Example:

```
MyWebJavaListClass.CreateHtml      PROCEDURE(*HtmlClass Target)

CurFont          HtmlFontClass
Feq               SIGNED,AUTO
Properties        ANY
CODE
  IF (Feq{PROP:hscroll})
    IF (Feq{PROP:vscroll})
      Properties = Properties & ',HVSCROLL'
    ELSE
      Properties = Properties & ',HSCROLL'
    END
  ELSE
    IF (Feq{PROP:vscroll})
      Properties = Properties & ',VSCROLL'
    END
  END
  Target.WriteAppletHeader(SELF.Feq, 'ClarionListBox', SELF.dx, SELF.dy)
  SELF.CreateColorParameters(Target, SELF.AutoSpotLink)
  Target.WriteAppletParameter('formatString', SELF.Feq{PROP:format})
  Target.WriteAppletParameter('Events', JavaEvents.GetEventString(SELF.EventActionQ,
Update:OnBrowser))
  Target.WriteAppletParameter('Properties', SUB(Properties, 2, -1))
  IF (Target.GetFontChanged(CurFont))
    Target.WriteAppletFontParameter(CurFont)
  END
  IF (SELF.AutoSpotLink)
    Target.WriteAppletOptParameter('AutoSpotLink', SELF.AutoSpotLink)
  END
  Target.WriteAppletFooter
  SELF.Started = FALSE
  SELF.UpdateState
```

## WebStringClass Methods

The WebStringClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebStringClass contains the methods listed below.

### SetBreakable (allow word wrap)

---

#### SetBreakable( *value* ), VIRTUAL

**SetBreakable** Allows long strings to wrap onto additional lines.

*value* A Boolean constant, variable, EQUATE or expression that indicates whether the string can wrap.

The **SetBreakable** method enables or disables the word wrap for the string. A *value* of one (1) enables word wrap; a *value* of zero (0) disables word wrap.

SetBreakable is a VIRTUAL method so that other base class methods can directly call the SetBreakable virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
IC:CurControl &= WebWindow.AddControl(?MyString)
IC:CurControl.SetBreakable(TRUE)
```

## WebHtmlStringClass Methods

The WebHtmlStringClass inherits all the methods of the WebStringClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHtmlStringClass contains the methods listed below.

### CreateCellContents (generate HTML text string)

**CreateCellContents**( *html object* ), VIRTUAL

#### CreateCellContents

Generates HTML code to represent a text string.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a text string.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method.

The proper positioning of each Web page control is accomplished by using HTML TABLEs and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)          !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## GetCellAttributes (return control attributes)

**GetCellAttributes**( *html object* ), **STRING**, **VIRTUAL**

**GetCellAttributes** Returns HTML to set attributes associated with the Web page control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **GetCellAttributes** method returns HTML to set attributes associated with the Web page control. If there are no associated attributes, **GetCellAttributes** returns a null string.

**GetCellAttributes** is a **VIRTUAL** method so that other base class methods can directly call the **GetCellAttributes** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **GetCellAttributes** method identifies the “centered” attribute if present, then calls the **PARENT.GetCellAttributes** method to return any other attributes.

The **LayoutHtmlClass.CreateHtml** method calls the **GetCellAttributes** method.

Return Data Type: **STRING**

Example:

```
MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)

CurCell      &LayoutCellClass,AUTO
Index         SIGNED,AUTO
NumRows       SIGNED,AUTO
Xindex        SIGNED,AUTO
Yindex        SIGNED,AUTO

CODE
NumRows = RECORDS(SELF.Rows)
Target.Write('<<TABLE' & SELF.Style & '>')
LOOP Yindex = 1 TO NumRows
  GET(SELF.Rows, Yindex)
  Target.Write('<<TR>')
  LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)
    GET(SELF.Rows.Columns, Xindex)
    CurCell &= SELF.Rows.Columns.Cell
    Target.Write('<<TD' & CurCell.GetCellAttributes() & '>')
    Target.WriteLine('<</TD>')
  END
  Target.Write('<</TR>')
END
Target.WriteLine('<</TABLE>')
```

See Also: **LayoutHtmlClass.CreateHtml**

## WebJavaStringClass Properties

The WebJavaStringClass inherits all the properties of the WebStringClass from which it is derived.

In addition to (or instead of) the inherited properties, the WebJavaStringClass contains the properties listed below.

### AutoSpotLink (hypertext links)

---

**AutoSpotLink**    **BYTE, PROTECTED**

The **AutoSpotLink** property indicates whether string text that appears to be a URL, email address, FTP site, etc., is implemented on the Web page as a “live” hypertext links or plain text. A value of one (1) generates a live link; a value of zero (0) generates plain text.

This property is **PROTECTED**, therefore, it can only be referenced by a WebJavaStringClass method, or a method in a class derived from WebJavaStringClass.

Implementation:        The SetAutoSpotLink method sets the value of the AutoSpotLink property.

See Also:                SetAutoSpotLink

### LastText (last transmitted value)

---

**LastText**                **ANY, PROTECTED**

The **LastText** property contains the last value sent to the Client browser for this control. The WebJavaStringClass object uses this property to minimize network traffic.

This property is **PROTECTED**, therefore, it can only be referenced by a WebJavaStringClass method, or a method in a class derived from WebJavaStringClass.

## WebJavaStringClass Methods

The WebJavaStringClass inherits all the methods of the WebStringClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebJavaStringClass contains the methods listed below.

### CreateHtml (write HTML for control and its attributes)

---

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml**      Writes the HTML code representing the STRING control and its attributes.

*html object*      The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the STRING control and its attributes.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:      The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

See Also:              LayoutHtmlClass.CreateHtml

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata PROCEDURE(*JslManagerClass Target)
Index   SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                         !get next control
  IF (SELF.Controls.ThisControl.GetVisible())       !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## CreateParams (write all string parameters)

---

### CreateParams( *html object* ), VIRTUAL

**CreateParams** Writes all the applet parameters for the STRING control.  
*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateParams** method writes all the applet parameters for the STRING control.

CreateParams is a VIRTUAL method so that other base class methods can directly call the CreateParams virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
WebJavaStringClass.CreateHtml PROCEDURE(*HtmlClass Target)
Height                SIGNED,AUTO
Width                 SIGNED,AUTO
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
GetPosition(SELF.Feq,,Width,Height)
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the STRING control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

GetAppletType is a VIRTUAL method so that other base class methods can directly call the GetAppletType virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetAppletType method returns 'ClarionStringControl.'

Return Data Type: STRING

Example:

```
WebJavaStringClass.CreateHtml PROCEDURE(*HtmlClass Target)
Height                SIGNED,AUTO
Width                 SIGNED,AUTO
CODE
IF (NOT SELF.OwnerWindow.AllowJava) THEN RETURN.
GetPosition(SELF.Feq,,Width,Height)
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

## Init (initialize WebJavaStringClass object)

---

**Init**( *control number*, *owner window* ), **VIRTUAL**

<b>Init</b>	Initializes the WebJavaStringClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate).
<i>owner window</i>	The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebJavaStringClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The Init method calls the WebControlClass.Init method, and sets the initial value of the LastText property equal to 'MagicValue<12>' so LastText doesn't coincidentally match previously blank text.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)

CODE
ASSERT(~NewControl &= NULL)
NewControl.Init(Feq, SELF)
SELF.AddControl(NewControl)
```

See Also: LastText, WebWindowClass.AddControl, WebControlClass.Init

## SetAutoSpotLink (set live hypertext links)

---

### SetAutoSpotLink( *value* ), VIRTUAL

**SetAutoSpotLink** Sets the automatic generation of live hypertext links.

*value* A Boolean constant, variable, EQUATE or expression assigned to the AutoSpotLink property.

The **SetAutoSpotLink** method enables or disables the automatic generation of live hypertext links for string text that appears to be a URL, email address, FTP site, etc. A *value* of one (1) enables live links; a *value* of zero (0) disables live links (generates plain text).

SetAutoSpotLink is a VIRTUAL method so that other base class methods can directly call the SetAutoSpotLink virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The SetAutoSpotLink method sets the AutoSpotLink property.

Example:

```
IC:CurControl &= WebWindow.AddControl(?MyString)
IC:CurControl.SetAutoSpotLink(TRUE)
```

See Also: [AutoSpotLink](#)

## WebCloseButtonClass Properties

The WebCloseButtonClass inherits all the properties of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited properties, the WebCloseButtonClass contains the properties listed below.

### Height (button height)

---

#### Height SIGNED

The **Height** property indicates height of the WINDOW control.

Implementation: The Init method sets the value of the dy property.

See Also: Init

### Width (button width)

---

#### Width SIGNED

The **Width** property indicates width of the WINDOW control.

Implementation: The Init method sets the value of the dx property.

See Also: Init

### X (button horizontal position)

---

#### X SIGNED

The **X** property indicates horizontal position of the WINDOW control.

Implementation: The Init method sets the value of the x property.

See Also: Init

### Y (button vertical position)

---

#### Y SIGNED

The **Y** property indicates vertical position of the WINDOW control.

Implementation: The Init method sets the value of the y property.

See Also: Init

## WebCloseButtonClass Methods

The WebCloseButtonClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebCloseButtonClass contains the methods listed below.

### CreateHtml (write HTML for control and its attributes)

---

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the control and its attributes.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the control and its attributes, including any image associated with the button.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The CreateHtml method sets the button's width, height, text, image, tooltip, and the browser action to take (close) when the end user presses the button.

The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```

MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem      &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)           !for all controls
  GET(SELF.Rows.Columns, Xindex)
  CurCell &= SELF.Rows.Columns.Cell
  NumItems = RECORDS(CurCell.Contents)
  Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
  LOOP Index = 1 TO NumItems                             !for all CELL items
    CurItem &= CurCell.GetItem(Index)                   !set WebControl object
    Style = CurItem.GetCellAttributes(Target)           !get control attributes
    IF (Style)
      Target.Write('<<P ' & Style & '>')             !begin HTML STYLE
    END
    CurItem.CreateHtml(Target)                          !write HTML control
    IF (Style)
      Target.Write('<</P>')                          !end HTML STYLE
    END
  END
  Target.WriteLine('<</TD>')                          !end HTML CELL
END

```

See Also:                    **LayoutHtmlClass.CreateHtml**

## CreateJsldata (update Web page control)

### CreateJsldata( *Jsl manager* ), VIRTUAL

**CreateJsldata** Writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

*Jsl manager* The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes Java Support Library (JSL) data to synchronize the Web page control with its corresponding WINDOW control.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The CreateJsldata method does nothing, because there is no corresponding WINDOW control for this Web page only control.

The WebWindowClass.CreateJsldata method calls the CreateJsldata method.

Example:

```
WebWindowClass.CreateJsldata  PROCEDURE(*JslManagerClass Target)

Index  SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)           !write data for all controls
  GET(SELF.Controls, Index)                         !get next control
  IF (SELF.Controls.ThisControl.GetVisible())      !if control is "visible"
    SELF.Controls.ThisControl.CreateJsldata(Target) !write its Jsl data
  END
END
```

See Also: WebWindowClass.CreateJsldata

## CreateParams (write all string parameters)

---

### CreateParams( *html object* ), VIRTUAL

**CreateParams** Writes all the applet parameters for the BUTTON control.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateParams** method writes all the applet parameters for the BUTTON control.

CreateParams is a VIRTUAL method so that other base class methods can directly call the CreateParams virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
WebCloseButtonClass.CreateHtml PROCEDURE(*HtmlClass Target)

CODE

IF SELF.OwnerWindow.AllowJava
    Target.WriteAppletHeader(SELF.Feq, SELF.GetAppletType(), SELF.width, SELF.height)
    SELF.CreateParams(Target)
    Target.WriteAppletFooter
ELSE
    Target.Write('<<INPUT TYPE=SUBMIT VALUE="Close"')
    Target.Write(SELF.GetNameAttribute(Target))
    Target.WriteLine('>')
END
```

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the **BUTTON** control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

**GetAppletType** is a **VIRTUAL** method so that other base class methods can directly call the **GetAppletType** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **GetAppletType** method returns 'ClarionImageButton.'

Return Data Type: **STRING**

Example:

```
WebCloseButtonClass.CreateHtml      PROCEDURE(*HtmlClass Target)

CODE

IF SELF.OwnerWindow.AllowJava
    Target.WriteAppletHeader(SELF.Feq, SELF.GetAppletType(), SELF.width, SELF.height)
    SELF.CreateParams(Target)
    Target.WriteAppletFooter
ELSE
    Target.Write('<<INPUT TYPE=SUBMIT VALUE="Close"')
    Target.Write(SELF.GetNameAttribute(Target))
    Target.WriteLine('>')
END
```

## GetCloneFeq (return button to mimic)

### GetCloneFeq, SIGNED, VIRTUAL

The **GetCloneFeq** method returns the control number of a similar control so the `WebCloseButtonClass` object can mimic its size.

`GetCloneFeq` is a `VIRTUAL` method so that other base class methods can directly call the `GetCloneFeq` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The `GetCloneFeq` method returns the `TBarBrwChange` (toolbar change button) control number.

The `Init` method calls the `GetCloneFeq` method.

Example:

```
WebCloseButtonClass.Init PROCEDURE(SIGNED Feq, *WebWindowBaseClass OwnerWindow)

CloneFeq          SIGNED

CODE

SELF.Feq = Feq
SELF.ParentFeq = FEQ:Toolbar
SELF.x = 9999
SELF.y = 2
SELF.dx = 16
SELF.dy = 14
SELF.OwnerWindow &= OwnerWindow

! Resize the close button to match the control returned from this function
CloneFeq = SELF.GetCloneFeq()
IF (CloneFeq{PROP:type} <> 0)
    GetPosition(CloneFeq,,SELF.y,SELF.dx,SELF.dy)
END
```

See Also:

**Init**

## GetPosition (get control coordinates)

---

**GetPosition**( *x*, *y*, *width*, *height* ), VIRTUAL

**GetPosition**

Returns the control coordinates.

*x*

A numeric variable to receive the control's horizontal position.

*y*

A numeric variable to receive the control's vertical position.

*width*

A numeric variable to receive the control's width.

*height*

A numeric variable to receive the control's height.

The **GetPosition** method returns the control's WINDOW coordinates. The `WebWindowClass` and the `WebControlClass` use this method to help set appropriate position-based parent/child relationships for the control. The `LayoutHtmlClass` uses this information to help position the control on the Web page.

`GetPosition` is a VIRTUAL method so that other base class methods can directly call the `GetPosition` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The `GetPosition` method returns the values of the `Height`, `Width`, `X`, and `Y`, properties.

See Also:

`Height`, `Width`, `X`, `Y`

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the close button should appear on the Web page. A return value of one (1) indicates the close button should appear on the Web page; a return value of zero (0) indicates the close button should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The close button always appears on the Web page because it has been explicitly requested.

Return Data Type: BYTE

Example:

```
WebControlListClass.AddControlsToLayout PROCEDURE |
    (*WebControlQueue Source, *LayoutHtmlClass Layout)

CurIndex          SIGNED,AUTO

CODE
LOOP CurIndex = 1 TO RECORDS(Source)
    GET(Source, CurIndex)
    IF (Source.ThisControl.GetVisible())
        Layout.Insert(Source.ThisControl)
    END
END
END
```

See Also: WebWindowClass.GetCreateClose, WebWindowClass.CreateClose

## Init (initialize WebCloseButtonClass object)

---

**Init**( *control number*, *owner window* ), **VIRTUAL**

<b>Init</b>	Initializes the WebCloseButtonClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate).
<i>owner window</i>	The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebCloseButtonClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The Init method sets the initial value of the Height, Width, X, and Y, properties. It attempts to provide an intelligent default size and position for the button.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)

CODE
ASSERT(~NewControl &= NULL)
NewControl.Init(Feq, SELF)
SELF.AddControl(NewControl)
```

See Also:

Height, Width, X, Y

## ResetControl (apply Web page button action)

### ResetControl( *submit item* ), VIRTUAL

**ResetControl** Applies the Web page button action to the Server.  
*submit item* The label of the SubmitItem object containing the information submitted by the Client browser for this control.

The **ResetControl** method applies the Web page button action to the Server. The ResetControl method takes Web page control information submitted by the Client browser, such as an event (EVENT:CloseWindow), and applies it to the Server.

ResetControl is a VIRTUAL method so that other base class methods can directly call the ResetControl virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The ResetControl method POSTs and EVENT:CloseWindow. There is no corresponding WINDOW control on the Server.

The WebWindowClass.TakeRequest method calls the ResetControl method.

Example:

```
MyWebWindowClass.TakeRequest  PROCEDURE
NextSubmit  &SubmitItemClass
CurFeq     SIGNED,AUTO
Index      SIGNED,AUTO
CODE
IF (SELF.Server.GetRequestedWholePage())
  LOOP Index = 1 TO RECORDS(SELF.Controls)           !preprocess all controls
    GET(SELF.Controls, Index)
    SELF.Controls.ThisControl.BeforeResetControl()   !preprocess a control
  END
END
LOOP                                           !synchronize all controls
NextSubmit &= SELF.Server.SetNextAction()         !set next submit item
IF (NextSubmit &= NULL) THEN BREAK.             !stop when finished
CurFeq = NextSubmit.Feq
IF (SELF.GetControl(CurFeq))                   !set corresponding control
  IF (NextSubmit.Event)                         !confirm event exists
    IF (NOT CurFeq{PROP:disable}) AND (NOT CurFeq{PROP:readonly})
      SELF.Controls.ThisControl.ResetControl(NextSubmit)!synchronize the control
  . . . . .
```

See Also: WebWindowClass.TakeRequest

## WebHotlinkClass Methods

The WebHotlinkClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebHotlinkClass contains the methods listed below.

### CreateCellContents (generate HTML hypertext link)

**CreateCellContents**( *html object* ), VIRTUAL

#### CreateCellContents

Generates HTML code to represent a hypertext link.

*html object*

The label of the HtmlClass object that writes the HTML code.

The **CreateCellContents** method generates HTML code to represent a hypertext link.

CreateCellContents is a VIRTUAL method so that other base class methods can directly call the CreateCellContents virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The WebControlClass.CreateHtml method calls the CreateCellContents method. The CreateCellContents method writes the appropriate <<A HREF...</A> construction.

The proper positioning of each Web page control is accomplished by using HTML TABLES and CELLS. CreateCellContents writes only the HTML code representing the control. The HTML code defining the TABLE and the CELL containing the control is written by other methods including LayoutHtmlClass.CreateHtml, WebControlClass.CreateCellHeader, and WebControlClass.CreateCellFooter.

Example:

```
WebControlClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.CreateCellHeader(Target)           !write HTML TABLE CELL header
SELF.CreateCellContents(Target)         !write HTML representing the control
SELF.CreateCellFooter(Target)          !write HTML TABLE CELL footer
```

See Also:

WebControlClass.CreateHtml, WebControlClass.CreateCellHeader, WebControlClass.CreateCellFooter, LayoutHtmlClass.CreateHtml

## *WebLiteralClass Properties*

The `WebLiteralClass` inherits all the properties of the `WebControlClass` from which it is derived.

In addition to (or instead of) the inherited properties, the `WebLiteralClass` contains the properties listed below.

### **Text (Web page text)**

---

<b>Text</b>	<b>CSTRING(1000)</b>
-------------	----------------------

The **Text** property contains the text to display on the Web page.

Implementation: The `Init` method sets the value of the `x` property.

See Also: `Init`

## WebLiteralClass Methods

The WebLiteralClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebLiteralClass contains the methods listed below.

### CreateHtml (write HTML for control)

#### CreateHtml( *html object* ), VIRTUAL

<b>CreateHtml</b>	Writes the HTML code representing the text.
<i>html object</i>	The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the text.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The LayoutHtmlClass.CreateHtml method calls the CreateHtml method.

Example:

```
MyLayoutHtmlClass.CreateHtml PROCEDURE(*HtmlClass Target)
CurItem      &HtmlItemClass,AUTO
!procedure data
CODE
!procedure code
LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)           !for all controls
GET(SELF.Rows.Columns, Xindex)
CurCell &= SELF.Rows.Columns.Cell
NumItems = RECORDS(CurCell.Contents)
Target.Write('<<TD' & CurCell.GetCellAttributes() & '>') !begin HTML CELL
LOOP Index = 1 TO NumItems                               !for all CELL items
CurItem &= CurCell.GetItem(Index)                      !set WebControl object
Style = CurItem.GetCellAttributes(Target)               !get control attributes
IF (Style)
Target.Write('<<P ' & Style & '>')                     !begin HTML STYLE
END
CurItem.CreateHtml(Target)                              !write HTML control
IF (Style)
Target.Write('<</P>')                                   !end HTML STYLE
END
END
Target.WriteLine('<</TD>')                               !end HTML CELL
END
```

See Also: LayoutHtmlClass.CreateHtml

## GetCellAttributes (return control attributes)

**GetCellAttributes**( *html object* ), **STRING**, **VIRTUAL**

**GetCellAttributes** Returns attributes associated with the text.

*html object*            The label of the HtmlClass object that writes the HTML code.

The **GetCellAttributes** method returns attributes associated with the text. If there are no associated attributes, **GetCellAttributes** returns a null string.

**GetCellAttributes** is a **VIRTUAL** method so that other base class methods can directly call the **GetCellAttributes** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:        The **GetCellAttributes** method always returns a null string because there are no attributes associated with a **WebLiteralClass** object.

The **LayoutHtmlClass.CreateHtml** method calls the **GetCellAttributes** method.

Return Data Type:     **STRING**

Example:

```
MyLayoutHtmlClass.CreateHtml  PROCEDURE(*HtmlClass Target)

CurCell      &LayoutCellClass,AUTO
Index         SIGNED,AUTO
NumRows      SIGNED,AUTO
Xindex       SIGNED,AUTO
Yindex       SIGNED,AUTO

CODE
NumRows = RECORDS(SELF.Rows)
Target.Write('<<TABLE' & SELF.Style & '>')
LOOP Yindex = 1 TO NumRows
  GET(SELF.Rows, Yindex)
  Target.Write('<<TR>')
  LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)
    GET(SELF.Rows.Columns, Xindex)
    CurCell &= SELF.Rows.Columns.Cell
    Target.Write('<<TD' & CurCell.GetCellAttributes() & '>')
    Target.WriteLine('<</TD>')
  END
  Target.Write('<</TR>')
END
END
Target.WriteLine('<</TABLE>')
```

See Also:                **LayoutHtmlClass.CreateHtml**

## WebNullControlClass Methods

The WebNullControlClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebNullControlClass contains the methods listed below.

### CreateHtml (write HTML for control)

---

#### CreateHtml( *html object* ), VIRTUAL

<b>CreateHtml</b>	Writes no HTML code.
<i>html object</i>	The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes no HTML code so that the WINDOW control does not appear on the Web page.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: WebWindowClass.SuppressControl

### CreateJsldata (update Web page control)

---

#### CreateJsldata( *Jsl manager* ), VIRTUAL

<b>CreateJsldata</b>	Writes no Java Support Library (JSL) data.
<i>Jsl manager</i>	The label of the JSLManagerClass object that sends data to Java applets on the Web page.

The **CreateJsldata** method writes no Java Support Library (JSL) data because the control does not appear on the Web page.

CreateJsldata is a VIRTUAL method so that other base class methods can directly call the CreateJsldata virtual method in a derived class. This lets you easily implement your own custom version of this method.

See Also: WebWindowClass.SuppressControl

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

GetAppletType is a VIRTUAL method so that other base class methods can directly call the GetAppletType virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetAppletType method returns 'NullApplet.'

Return Data Type: STRING

## GetCellAttributes (return control attributes)

**GetCellAttributes**( *html object* ), **STRING**, **VIRTUAL**

**GetCellAttributes** Returns a null string.

*html object*            The label of the HtmlClass object that writes the HTML code.

The **GetCellAttributes** method returns a null string because the control does not appear on the Web page..

GetCellAttributes is a VIRTUAL method so that other base class methods can directly call the GetCellAttributes virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type:        **STRING**

Example:

```
MyLayoutHtmlClass.CreateHtml  PROCEDURE(*HtmlClass Target)

CurCell            &LayoutCellClass,AUTO
Index                SIGNED,AUTO
NumRows             SIGNED,AUTO
Xindex               SIGNED,AUTO
Yindex               SIGNED,AUTO

CODE
NumRows = RECORDS(SELF.Rows)
Target.Write('<<TABLE' & SELF.Style & '>')
LOOP Yindex = 1 TO NumRows
  GET(SELF.Rows, Yindex)
  Target.Write('<<TR>')
  LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)
    GET(SELF.Rows.Columns, Xindex)
    CurCell &= SELF.Rows.Columns.Cell
    Target.Write('<<TD' & CurCell.GetCellAttributes() & '>')
    Target.WriteLine('<</TD>')
  END
  Target.Write('<</TR>')
END
Target.WriteLine('<</TABLE>')
```

See Also:                **WebWindowClass.SuppressControl**

## GetIsChild (return family identity)

---

**GetIsChild**( *parent control* [, *control type*] ), **SIGNED, VIRTUAL**

**GetIsChild** Returns a value indicating whether this WebControlClass object is a visible child control of the specified *parent control* and *control type*.

*parent control* An integer constant, variable, EQUATE, or expression containing the parent control's control number, or zero (0) if the WINDOW is the parent.

*control type* An integer constant, variable, EQUATE, or expression containing the type of control sought. If omitted, any control type is valid.

The **GetIsChild** method returns a value indicating whether this WebControlClass object is a visible child control of the specified *parent control* and *control type*. If this WebControlClass object meets the specified criteria it returns its own control number (Feq property), otherwise it returns zero (0) to indicate it does not meet the specified criteria.

GetIsChild is a VIRTUAL method so that other base class methods can directly call the GetIsChild virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetIsChild method returns zero (0) because a WebNullControlClass object has no parent.

EQUATEs for the *control type* parameter are declared in EQUATES.CLW. Each control type EQUATE is prefixed with CREATE:.

Return Data Type: SIGNED

See Also: WebWindowClass.SuppressControl

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating the control should not appear on the Web page because the `WebNullControlClass` object's purpose is to suppress the control from the Web page.

`GetVisible` is a `VIRTUAL` method so that other base class methods can directly call the `GetVisible` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Return Data Type: **BYTE**

See Also: `WebWindowClass.SuppressControl`

# WEB AREA CLASSES

<b>Overview</b>	<b>409</b>
WebAreaClass Concepts .....	409
Relationship to Other Internet Builder Classes .....	409
Internet Connect Template Implementation .....	410
Source Files .....	410
<b>WebAreaClass Properties</b>	<b>411</b>
Background (area background color) .....	411
BackImage (area wallpaper) .....	411
LocalFont (area font information) .....	411
<b>WebAreaClass Methods</b>	<b>412</b>
GetBackgroundColor (return background color) .....	412
GetCellAttributes (return area attributes) .....	413
GetFont (add font information) .....	414
GetVisible (return control status flag) .....	415
Init (initialize the WebAreaClass object) .....	416
Kill (shut down the WebAreaClass object) .....	417
PushFont (implement area font) .....	418
SetBackground (set Web page area background) .....	419
SetFont .....	420
SetParentDefaults (confirm parent) .....	421
<b>WebCaptionClass Properties</b>	<b>422</b>
Alignment (text justification) .....	422
<b>WebCaptionClass Methods</b>	<b>423</b>
CreateHtml (write HTML for caption and its attributes) .....	423
GetCellAttributes (return caption attributes) .....	424
GetPosition (get control coordinates) .....	425
GetText (return caption text) .....	426
GetVisible (return control status flag) .....	427
Init (initialize the WebCaptionClass object) .....	428
<b>WebMenuBarClass Methods</b>	<b>429</b>
CreateHtml (write HTML for menubar and its children) .....	429
GetPosition (get control coordinates) .....	430
GetVisible (return control status flag) .....	431
Init (initialize the WebMenuBarClass object) .....	432

<b>WebToolBarClass Methods</b>	<b>433</b>
CreateHtml (write HTML for toolbar and its children) .....	433
GetAppletType (return applet type) .....	434
GetPosition (get control coordinates) .....	435
GetVisible (return control status flag) .....	436
<b>WebClientAreaClass Methods</b>	<b>437</b>
CreateHtml (write HTML for client area and its children) .....	437
GetBackgroundColor (return background color) .....	438
GetPosition (get control coordinates) .....	439

## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### WebAreaClass Concepts

---

The WebAreaClass is the foundation for several derived classes (WebCaptionClass, WebMenubarClass, WebToolbarClass, and WebClientAreaClass). The derived class objects implement areas or sections of a Web page that correspond to specific areas of a Clarion WINDOW or APPLICATION.

Some of the WebAreaClass objects directly correspond to WINDOW controls; for example, the WebToolbarClass represents a TOOLBAR and the WebMenubarClass represents a MENUBAR. However, the other WebAreaClass objects represent pseudo-controls. These are discrete portions of the WINDOW, such as the titlebar and the client area, that are not separate Clarion controls, but are more easily handled as separate controls within the Web page context.

### Relationship to Other Internet Builder Classes

---

All the classes described in this chapter are derived from the WebControlClass; therefore, the information and documentation pertaining to the WebControlClass is directly applicable to these derived classes. Please see *Web Control Class* for more information.

#### WebControlClass

The WebAreaClass is derived from the WebControlClass. Therefore, its derived class objects are WebControlClass objects—plus a little more. Their purpose is to generate HTML/JavaScript to implement Web page components that look and act like their corresponding Windows window components.

#### Derived Classes

WebCaptionClass, WebMenubarClass, WebToolbarClass, and WebClientAreaClass are derived from WebAreaClass.

## Internet Connect Template Implementation

---

The WebWindowClass creates and manages instances of the classes derived from the WebAreaClass as needed to generate HTML for each area of the WINDOW. Therefore, the template generated code does not directly reference WebAreaClass objects, or its derived class objects.

## Source Files

---

The WebAreaClass source code is installed by default to the \LIBSRC folder. The WebAreaClass declarations are in the following .INC files. The corresponding method definitions are in the corresponding .CLW file.

ICCNTRLS.INC	WebAreaClass
	WebCaptionClass
	WebMenubarClass
	WebToolbarClass
	WebClientAreaClass

## WebAreaClass Properties

The WebAreaClass inherits all the properties of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited properties, the WebAreaClass contains the properties listed below.

### Background (area background color)

---

Background	LONG(COLOR:None)
------------	------------------

The **Background** property indicates the background color of the Web page area.

Implementation: The SetBackground method sets the value of the Background property.

See Also: SetBackground

### BackImage (area wallpaper)

---

BackImage	ANY
-----------	-----

The **BackImage** property indicates the background image (wallpaper) to apply to the Web page area.

Implementation: The SetBackground method sets the value of the BackImage property.

See Also: SetBackground

### LocalFont (area font information)

---

LocalFont	&HtmlFontClass
-----------	----------------

The **LocalFont** property contains the area's font information for use by methods such as PushFont, PopFont, and SetFont. The font information includes the typeface, size, color, and style.

Controls within the area inherit the area's font information, unless specifically overridden by the control.

See Also: GetFont, PushFont, SetFont, WebControlClass.PopFont

## WebAreaClass Methods

The WebAreaClass inherits all the methods of the WebControlClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebAreaClass contains the methods listed below.

### GetBackgroundColor (return background color)

**GetBackgroundColor( [*default color* ]), LONG, VIRTUAL**

#### GetBackgroundColor

Returns the background color of the Web page area.

*default color* An integer variable, constant, EQUATE, or expression containing the color to return if there is no background color. If omitted, *default color* defaults to Color:None.

The **GetBackgroundColor** method returns the background color of the Web page area.

GetBackgroundColor is a VIRTUAL method so that other base class methods can directly call the GetBackgroundColor virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetBackgroundColor method returns the COLOR attribute of the WINDOW control, if there is one. Otherwise, it returns the background color inherited from the Web page. EQUATES for the *default color* parameter are declared in \LIBSRC\EQUATES.CLW.

The CreateColorParameters method calls GetBackgroundColor.

Return Data Type: LONG

Example:

```
MyWebControlClass.CreateColorParameters PROCEDURE(*HtmlClass Target, BYTE AutoSpotLink)
ForeColor          LONG,AUTO
BackColor          LONG,AUTO
CODE
GETFONT(SELF.Feq,, ForeColor)
BackColor = SELF.GetBackgroundColor()
IF (ForeColor <> 0) AND (ForeColor <> COLOR:None)
    Target.WriteAppletParameter('ForeColor', IC:RGB(ForeColor))
END
IF (BackColor <> COLOR:None)
    Target.WriteAppletParameter('BackColor', IC:RGB(BackColor))
END
```

See Also: CreateColorParameters

## GetCellAttributes (return area attributes)

**GetCellAttributes**( *html object* ), **STRING**, **VIRTUAL**

**GetCellAttributes** Returns attributes associated with the Web page area.

*html object*            The label of the HtmlClass object that writes the HTML code.

The **GetCellAttributes** method returns attributes associated with the Web page area or control. If there are no associated attributes, **GetCellAttributes** returns a null string.

**GetCellAttributes** is a **VIRTUAL** method so that other base class methods can directly call the **GetCellAttributes** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:        The **GetCellAttributes** method returns a text string containing the HTML definition of the area's background color, background image (wallpaper) and font information.

The **LayoutHtmlClass.CreateHtml** method calls the **GetCellAttributes** method.

Return Data Type:      **STRING**

Example:

```
MyLayoutHtmlClass.CreateHtml  PROCEDURE(*HtmlClass Target)

CurCell      &LayoutCellClass,AUTO
Index         SIGNED,AUTO
NumRows      SIGNED,AUTO
Xindex       SIGNED,AUTO
Yindex       SIGNED,AUTO

CODE
NumRows = RECORDS(SELF.Rows)
Target.Write('<<TABLE' & SELF.Style & '>')
LOOP Yindex = 1 TO NumRows
  GET(SELF.Rows, Yindex)
  Target.Write('<<TR>')
  LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)
    GET(SELF.Rows.Columns, Xindex)
    CurCell &= SELF.Rows.Columns.Cell
    Target.Write('<<TD' & CurCell.GetCellAttributes() & '>')
    Target.WriteLine('<</TD>')
  END
  Target.Write('<</TR>')
END
END
Target.WriteLine('<</TABLE>')
```

See Also:                **LayoutHtmlClass.CreateHtml**

## GetFont (add font information)

---

### GetFont( *font object* )

**GetFont** Adds the control's font information to the *font object*.  
*font object* The label of the HtmlFontClass object that manages font information.

The **GetFont** method adds the control's font information to the *font object* for subsequent use by other methods such as `HtmlClass.WriteAppletFontParameter`. The font information includes the typeface, size, color, and style.

Implementation: The `GetFont` method returns the values in the `LocalFont` property.

Example:

```
MyWebToolBarClass.CreateHtml      PROCEDURE(*HtmlClass Target)
CurFont      HtmlFontClass
!procedure data
CODE
SELF.PushFont(Target)
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.GetFont(CurFont)
Target.WriteAppletFontParameter(CurFont)
Target.WriteAppletFooter
SELF.PopFont(Target)
```

See Also: [LocalFont](#)

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the area should appear on the Web page. A return value of one (1) indicates the area should appear on the Web page; a return value of zero (0) indicates the area should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

**Implementation:** An area may be omitted from the Web page for several reasons: it may be disabled or hidden, or it may be explicitly omitted. For example, a caption or toolbar may be explicitly excluded from the Web page. See `WebWindowClass.CreateCaption` and `WebWindowClass.CreateToolbar`.

**Return Data Type:** BYTE

**Example:**

```
WebWindowClass.CreateJsldata PROCEDURE(*Js1ManagerClass Target)

Index          SIGNED,AUTO

CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.GetVisible())
    SELF.Controls.ThisControl.CreateJsldata(Target)
  END
END
```

## Init (initialize the WebAreaClass object)

---

**Init**( *control number*, *owner window object* ), **VIRTUAL**

<b>Init</b>	Initializes the WebAreaClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate). For areas that have no corresponding control, use any number that is not already in use.
<i>owner window object</i>	The label of the WebWindowClass object the control belongs to.

The **Init** method initializes the WebAreaClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The Init method sets the initial value of the Background, BackImage, Feq, OwnerWindow, and ParentFeq properties.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)

CODE
ASSERT(~NewControl &= NULL)
NewControl.Init(Feq, SELF)
SELF.AddControl(NewControl)
```

See Also: Background, BackImage, Feq, OwnerWindow, ParentFeq, WebWindowClass.AddControl

## Kill (shut down the WebAreaClass object)

---

### Kill, VIRTUAL

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Kill is a VIRTUAL method so that other base class methods can directly call the Kill virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The WebWindowClass.Kill method calls the Kill method.

Example:

```
MyWebWindowClass.Kill                   PROCEDURE
Index                   SIGNED,AUTO
CODE
IF (~SELF.Controls &= NULL)
  LOOP Index = 1 TO RECORDS(SELF.Controls)
    GET(SELF.Controls, Index)
    ASSERT(~SELF.Controls.ThisControl &= NULL)
    SELF.Controls.ThisControl.Kill
  END
DISPOSE(SELF.Controls)
END
```

See Also:               WebWindowClass.Kill

## PushFont (implement area font)

---

### PushFont( *html object* ), VIRTUAL

<b>PushFont</b>	Implements the area's font in the HTML generated for the area.
<i>html object</i>	The label of the HtmlClass object that writes the HTML code.

The **PushFont** method implements the area's font in the HTML generated for the area. Appropriate use of PushFont and PopFont allow child controls to correctly inherit or override parent font information. Appropriate use simply means pairing a call to PushFont with a subsequent call to PopFont.

PushFont is a VIRTUAL method so that other base class methods can directly call the PopFont virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: If the area has no font information, PushFont implements the WINDOW's font information. The PushFont method calls the HtmlClass.PushFont method.

Example:

```
WebMenuBarClass.CreateHtml      PROCEDURE(*HtmlClass Target)
  
CODE
SELF.PushFont(Target)
SELF.OwnerWindow.CreateChildHtml(Target, FEQ:MenuBar)
SELF.PopFont(Target)
```

See Also: [WebControlClass.PopFont](#)

## SetBackground (set Web page area background)

---

**SetBackground( [color] [,image] )**

**SetBackground** Sets the Web page area background color and image.

*color* An integer constant, variable, EQUATE, or expression indicating the area color. The area's child controls inherit this color unless a specific background color is set for the control.

*image* A string constant, variable, EQUATE, or expression containing a filename. The WebAreaClass object displays (tiles) the image in the specified file as the background to the Web page area.

The **SetBackground** method sets the Web page area background color and image. Area child controls inherit the background color unless overridden. Area child controls do not inherit the image.

Implementation: The SetBackground method sets the Background property and the BackImage property. The GetBackgroundColor method returns the value of the Background property.

Example:

```
PrepareProcedure ROUTINE
FilesOpened = True
OPEN(AppFrame)
WindowOpened=True
WebWindow.Init(WebServer, HtmlManager, AppFrame{PROP:text} & ' (Main)')
WebWindow.MenuBarType = PROP:above
WebWindow.CreateCaption = 1
WebWindow.AddControl(FEQ:Caption, WebCaption)
WebCaption.SetBackground(-1, '')
WebCaption.SetFont('', 0, -1)
```

See Also: Background, BackImage, WebControlClass.GetBackgroundColor

## SetFont

**SetFont**( [*typeface*] [,*size*] [,*color*] [,*style*] )

<b>SetFont</b>	Sets the Web page area font information.
<i>typeface</i>	A string constant, variable, EQUATE, or expression indicating the typeface for text in the Web page area. The area's child controls inherit this typeface unless overridden. If omitted, the browser's default typeface is used.
<i>size</i>	An integer constant, variable, EQUATE, or expression indicating the text point size. The area's child controls inherit this size unless overridden. If omitted, <i>size</i> defaults to zero (0).
<i>color</i>	An integer constant, variable, EQUATE, or expression indicating the text color. The area's child controls inherit this color unless overridden. If omitted, <i>color</i> defaults to COLOR:None.
<i>style</i>	An integer constant, variable, EQUATE, or expression indicating the text style, such as bold or italic. The area's child controls inherit this style unless overridden. If omitted, <i>style</i> defaults to zero (0).

The **SetFont** method sets the Web page area font information, including typeface, size, color, and style. Area child controls inherit the font information unless overridden.

When setting font information you should consider whether the typeface you specify is likely to reside on the Client machine. The Client browser cannot display a typeface it doesn't have, but will usually substitute another typeface.

Implementation: The **SetFont** method sets the **LocalFont** property. The **PushFont** method implements the font information on the Web page area. The **PopFont** method restores pre-**PushFont** font information.

Example:

```
WebCaptionClass.Init PROCEDURE(SIGNED Fc, *WebWindowBaseClass OwnerWindow)
```

```
CODE
PARENT.Init(Fc, OwnerWindow)
SELF.Background = COLOR:Navy
SELF.SetFont(, ,COLOR:White)
```

See Also: **LocalFont**, **PushFont**, **WebContolClass.PopFont**

## SetParentDefaults (confirm parent)

**SetParentDefaults**( *potential parent*, *coordinates* ), **VIRTUAL**

**SetParentDefaults** Confirms this control's boundaries fall entirely within the *potential parent* boundaries.

*potential parent* The label of a WebControlClass object.

*coordinates* The label of a structure containing the coordinates of the *potential parent*.

The **SetParentDefaults** method confirms this control's boundaries fall entirely within the *potential parent* boundaries, and if so, sets the ParentFeq property to reflect the parent/child relationship.

SetParentDefaults is a VIRTUAL method so that other base class methods can directly call the SetParentDefaults virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The SetParentDefaults method does nothing because a Web page area is never the child of another control. It is always the child of the window.

The *coordinates* parameter names a GROUP with the same structure as the Rect GROUP declared in ICWINDOW.INC as follows:

Rect	GROUP,TYPE
x	SIGNED
y	SIGNED
width	SIGNED
height	SIGNED
	END

Example:

```
WebControlListClass.SetParentDefaults PROCEDURE|
  (*WebControlQueue Source, *WebControlClass Other, *Rect ParentPos)

CurIndex          SIGNED,AUTO

CODE
LOOP CurIndex = 1 TO RECORDS(Source)
  GET(Source, CurIndex)
  Source.ThisControl.SetParentDefaults(Other, ParentPos)
END
```

See Also:

WebWindowClass.SetParentDefaults

## *WebCaptionClass Properties*

The WebCaptionClass inherits all the properties of the WebAreaClass from which it is derived.

In addition to (or instead of) the inherited properties, the WebCaptionClass contains the properties listed below.

### **Alignment (text justification)**

---

Alignment	SIGNED
-----------	--------

The **Alignment** property indicates how to horizontally align or justify the caption area text. The WebCaptionClass object uses this property to generate HTML to properly position the caption text.

The Alignment property defaults to zero(0) which results in centered text.

Implementation:

EQUATEs for the Alignment property are declared in PROPERTY.CLW as follows.

PROP:center	EQUATE(7C06H)
PROP:left	EQUATE(7C08H)
PROP:right	EQUATE(7C0CH)

## WebCaptionClass Methods

The WebCaptionClass inherits all the methods of the WebAreaClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebCaptionClass contains the methods listed below.

### CreateHtml (write HTML for caption and its attributes)

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the titlebar and its attributes.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the titlebar and its attributes.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: Attributes include text, font information, and text alignment.

The LayoutHtmlClass.CreateHtml method calls the WebAreaClass.CreateHtml method.

Example:

```
MyWebCaptionClass.CreateHtml      PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.PushFont(Target)
Target.WriteControlHeader
Target.Write(SELF.GetQuotedText())
Target.WriteControlFooter
SELF.PopFont(Target)
```

See Also: LayoutHtmlClass.CreateHtml

## GetCellAttributes (return caption attributes)

**GetCellAttributes**( *html object* ), **STRING**, **VIRTUAL**

**GetCellAttributes** Returns HTML to set any attributes associated with the Web page caption.

*html object* The label of the HtmlClass object that writes the HTML code.

The **GetCellAttributes** method returns HTML to set any attributes associated with the Web page caption.

GetCellAttributes is a VIRTUAL method so that other base class methods can directly call the GetCellAttributes virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetCellAttributes method returns the HTML specifying text alignment, plus any attributes inherited from the WINDOW.

The LayoutHtmlClass.CreateHtml method calls the GetCellAttributes method.

Return Data Type: **STRING**

Example:

```
MyLayoutHtmlClass.CreateHtml  PROCEDURE(*HtmlClass Target)

CurCell      &LayoutCellClass,AUTO
Index         SIGNED,AUTO
NumRows      SIGNED,AUTO
Xindex       SIGNED,AUTO
Yindex       SIGNED,AUTO

CODE
NumRows = RECORDS(SELF.Rows)
Target.Write('<<TABLE' & SELF.Style & '>')
LOOP Yindex = 1 TO NumRows
  GET(SELF.Rows, Yindex)
  Target.Write('<<TR>')
  LOOP Xindex = 1 TO RECORDS(SELF.Rows.Columns)
    GET(SELF.Rows.Columns, Xindex)
    CurCell &= SELF.Rows.Columns.Cell
    Target.Write('<<TD' & CurCell.GetCellAttributes() & '>')
    Target.WriteLine('<</TD>')
  END
  Target.Write('<</TR>')
END
Target.WriteLine('<</TABLE>')
```

See Also: [LayoutHtmlClass.CreateHtml](#)

## GetPosition (get control coordinates)

**GetPosition**( *x*, *y*, *width*, *height* ), **VIRTUAL**

<b>GetPosition</b>	Returns the caption's Web page coordinates.
<i>x</i>	A numeric variable to receive the control's horizontal position.
<i>y</i>	A numeric variable to receive the control's vertical position.
<i>width</i>	A numeric variable to receive the control's width.
<i>height</i>	A numeric variable to receive the control's height.

The **GetPosition** method returns the caption's Web page coordinates. The `LayoutHtmlClass` uses this information to position the caption on the Web page.

`GetPosition` is a **VIRTUAL** method so that other base class methods can directly call the `GetPosition` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
LayoutHtmlClass.Insert PROCEDURE(*HtmlItemClass NewItem)

CurCell      &LayoutCellClass,AUTO
dx            SIGNED,AUTO
dy            SIGNED,AUTO
x             SIGNED,AUTO
y             SIGNED,AUTO
Xpos         SIGNED,AUTO
Ypos         SIGNED,AUTO
IsNew        BYTE,AUTO
CODE
NewItem.GetPosition(x, y, dx, dy)
Ypos = SELF.RangeY.AddPoint(y, dy, IsNew, SELF.SnapY)
IF (IsNew) THEN SELF.AddRow(Ypos).
Xpos = SELF.RangeX.AddPoint(x, dx, IsNew, SELF.SnapX)
IF (IsNew) THEN SELF.AddColumn(Xpos).
CurCell &= SELF.SetCell(Xpos, Ypos)
CurCell.Contents.Item &= NewItem
ADD(CurCell.Contents)
IF (dx > CurCell.dx)
    CurCell.dx = dx;
END
IF (dy > CurCell.dy)
    CurCell.dy = dy;
END
```

## GetText (return caption text)

---

### GetText, STRING, VIRTUAL

The **GetText** method returns the caption text.

GetText is a VIRTUAL method so that other base class methods can directly call the GetText virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetQuotedText method calls the GetText method to retrieve the text. GetText returns PROP:Text for the WINDOW.

Return Data Type: STRING

Example:

```
WebControlClass.GetQuotedText      FUNCTION
CODE
RETURN IC:QuoteText(SELF.GetText(), IC:RESET:Text)
```

See Also: GetHasHotKey, GetQuotedText

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the caption should appear on the Web page. A return value of one (1) indicates the caption should appear on the Web page; a return value of zero (0) indicates the caption should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:      GetVisible returns one (1) if the WINDOW has titlebar text; zero (0) if it has no titlebar text.

Return Data Type:      BYTE

Example:

```
WebWindowClass.CreateJs1Data      PROCEDURE(*Js1ManagerClass Target)
Index                              SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.GetVisible())
    SELF.Controls.ThisControl.CreateJs1Data(Target)
  END
END
```

## Init (initialize the WebCaptionClass object)

---

**Init**( *control number*, *owner window object* ), **VIRTUAL**

### **Init**

Initializes the WebCaptionClass object.

*control number*

An integer constant, variable, EQUATE, or expression containing the control number (field equate). Since the caption has no control number, you can use any number that is not already in use.

*owner window object*

The label of the WebWindowClass object the caption belongs to.

The **Init** method initializes the WebCaptionClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The Init method calls the WebAreaClass.Init method, then sets the initial value of the Background property.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)
```

```
CODE  
ASSERT(~NewControl &= NULL)  
NewControl.Init(Feq, SELF)  
SELF.AddControl(NewControl)
```

See Also:

Background, WebWindowClass.AddControl, WebAreaClass.Init

## WebMenuBarClass Methods

The WebMenuBarClass inherits all the methods of the WebAreaClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebMenuBarClass contains the methods listed below.

### CreateHtml (write HTML for menubar and its children)

---

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml**      Writes the HTML code representing the MENUBAR and its children.

*html object*      The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the MENUBAR and its children.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The LayoutHtmlClass.CreateHtml method calls the WebAreaClass.CreateHtml method.

The CreateHtml method calls the WebWindowClass.CreateChildHtml method to write HTML for its children (MENU and ITEM controls).

Example:

```
WebMenuBarClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.PushFont(Target)
SELF.OwnerWindow.CreateChildHtml(Target, FEQ:MenuBar)
SELF.PopFont(Target)
```

See Also:

LayoutHtmlClass.CreateHtml, WebWindowClass.CreateChildHtml

## GetPosition (get control coordinates)

**GetPosition**( *x, y, width, height* ), VIRTUAL

<b>GetPosition</b>	Returns the menubar's Web page coordinates.
<i>x</i>	A numeric variable to receive the control's horizontal position.
<i>y</i>	A numeric variable to receive the control's vertical position.
<i>width</i>	A numeric variable to receive the control's width.
<i>height</i>	A numeric variable to receive the control's height.

The **GetPosition** method returns the menubar's Web page coordinates. The `LayoutHtmlClass` uses this information to position the menubar on the Web page.

`GetPosition` is a VIRTUAL method so that other base class methods can directly call the `GetPosition` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
LayoutHtmlClass.Insert PROCEDURE(*HtmlItemClass NewItem)

CurCell      &LayoutCellClass,AUTO
dx            SIGNED,AUTO
dy            SIGNED,AUTO
x             SIGNED,AUTO
y             SIGNED,AUTO
Xpos         SIGNED,AUTO
Ypos         SIGNED,AUTO
IsNew        BYTE,AUTO
CODE
NewItem.GetPosition(x, y, dx, dy)
Ypos = SELF.RangeY.AddPoint(y, dy, IsNew, SELF.SnapY)
IF (IsNew) THEN SELF.AddRow(Ypos).
Xpos = SELF.RangeX.AddPoint(x, dx, IsNew, SELF.SnapX)
IF (IsNew) THEN SELF.AddColumn(Xpos).
CurCell &= SELF.SetCell(Xpos, Ypos)
CurCell.Contents.Item &= NewItem
ADD(CurCell.Contents)
IF (dx > CurCell.dx)
    CurCell.dx = dx;
END
IF (dy > CurCell.dy)
    CurCell.dy = dy;
END
```

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the menubar should appear on the Web page. A return value of one (1) indicates the menubar should appear on the Web page; a return value of zero (0) indicates the menubar should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:      GetVisible calls the WebWindowClass.GetShowMenubar method.

Return Data Type:    BYTE

Example:

```
WebWindowClass.CreateJsldata   PROCEDURE(*JsIManagerClass Target)
Index                           SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.GetVisible())
    SELF.Controls.ThisControl.CreateJsldata(Target)
  END
END
```

See Also:            WebWindowClass.GetShowMenubar

## Init (initialize the WebMenuBarClass object)

---

**Init**( *control number, owner window object* ), **VIRTUAL**

<b>Init</b>	Initializes the WebMenuBarClass object.
<i>control number</i>	An integer constant, variable, EQUATE, or expression containing the control number (field equate). Since the caption has no control number, you can use any number that is not already in use.
<i>owner window object</i>	The label of the WebWindowClass object the menubar belongs to.

The **Init** method initializes the WebMenuBarClass object.

Init is a VIRTUAL method so that other base class methods can directly call the Init virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The Init method calls the WebAreaClass.Init method, then sets the initial value of the Background property.

The WebWindowClass.AddControl method calls the Init method.

Example:

```
WebWindowClass.AddControl PROCEDURE(SIGNED Feq, *WebControlClass NewControl)

CODE
ASSERT(~NewControl &= NULL)
NewControl.Init(Feq, SELF)
SELF.AddControl(NewControl)
```

See Also: Background, WebWindowClass.AddControl, WebAreaClass.Init

## WebToolBarClass Methods

The WebToolBarClass inherits all the methods of the WebAreaClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebToolBarClass contains the methods listed below.

### CreateHtml (write HTML for toolbar and its children)

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the TOOLBAR and its children.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the TOOLBAR and its children.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The LayoutHtmlClass.CreateHtml method calls the WebAreaClass.CreateHtml method.

The CreateHtml method calls the WebWindowClass.CreateChildHtml method to write HTML for its child controls.

Example:

```
WebToolBarClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.PushFont(Target)
SELF.OwnerWindow.CreateChildHtml(Target, FEQ:ToolBar)
SELF.PopFont(Target)
```

See Also: LayoutHtmlClass.CreateHtml, WebWindowClass.CreateChildHtml

## GetAppletType (return applet type)

---

### GetAppletType, STRING, VIRTUAL

The **GetAppletType** method returns the applet type for the TOOLBAR control. The type corresponds to an applet defined in the Java Support Library. See *Jsl Manager Class* for more information on the Java Support Library.

GetAppletType is a VIRTUAL method so that other base class methods can directly call the GetAppletType virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:       The GetAppletType method returns 'ClarionContainer.'

Return Data Type:     STRING

Example:

```
MyWebToolbarClass.CreateHtml       PROCEDURE(*HtmlClass Target)
CurFont                    HtmlFontClass
!procedure data
  CODE
  SELF.PushFont(Target)
  Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
  SELF.GetFont(CurFont)
  Target.WriteAppletFontParameter(CurFont)
  Target.WriteAppletFooter
  SELF.PopFont(Target)
```

## GetPosition (get control coordinates)

**GetPosition**( *x, y, width, height* ), VIRTUAL

<b>GetPosition</b>	Returns the toolbar's Web page coordinates.
<i>x</i>	A numeric variable to receive the control's horizontal position.
<i>y</i>	A numeric variable to receive the control's vertical position.
<i>width</i>	A numeric variable to receive the control's width.
<i>height</i>	A numeric variable to receive the control's height.

The **GetPosition** method returns the toolbar's Web page coordinates. The `LayoutHtmlClass` uses this information to position the toolbar on the Web page.

`GetPosition` is a VIRTUAL method so that other base class methods can directly call the `GetPosition` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
LayoutHtmlClass.Insert PROCEDURE(*HtmlItemClass NewItem)

CurCell      &LayoutCellClass,AUTO
dx            SIGNED,AUTO
dy            SIGNED,AUTO
x             SIGNED,AUTO
y             SIGNED,AUTO
Xpos          SIGNED,AUTO
Ypos          SIGNED,AUTO
IsNew        BYTE,AUTO
CODE
NewItem.GetPosition(x, y, dx, dy)
Ypos = SELF.RangeY.AddPoint(y, dy, IsNew, SELF.SnapY)
IF (IsNew) THEN SELF.AddRow(Ypos).
Xpos = SELF.RangeX.AddPoint(x, dx, IsNew, SELF.SnapX)
IF (IsNew) THEN SELF.AddColumn(Xpos).
CurCell &= SELF.SetCell(Xpos, Ypos)
CurCell.Contents.Item &= NewItem
ADD(CurCell.Contents)
IF (dx > CurCell.dx)
    CurCell.dx = dx;
END
IF (dy > CurCell.dy)
    CurCell.dy = dy;
END
```

## GetVisible (return control status flag)

---

### GetVisible, BYTE, VIRTUAL

The **GetVisible** method returns a value indicating whether the toolbar should appear on the Web page. A return value of one (1) indicates the toolbar should appear on the Web page; a return value of zero (0) indicates the toolbar should not appear.

GetVisible is a VIRTUAL method so that other base class methods can directly call the GetVisible virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:      GetVisible calls the WebWindowClass.GetShowToolbar method.

Return Data Type:    BYTE

Example:

```
WebWindowClass.CreateJsldata    PROCEDURE(*JsIManagerClass Target)
Index                            SIGNED,AUTO
CODE
LOOP Index = 1 TO RECORDS(SELF.Controls)
  GET(SELF.Controls, Index)
  IF (SELF.Controls.ThisControl.GetVisible())
    SELF.Controls.ThisControl.CreateJsldata(Target)
  END
END
```

See Also:            WebWindowClass.GetShowToolbar

## WebClientAreaClass Methods

The WebClientAreaClass inherits all the methods of the WebAreaClass from which it is derived.

In addition to (or instead of) the inherited methods, the WebClientAreaClass contains the methods listed below.

### CreateHtml (write HTML for client area and its children)

#### CreateHtml( *html object* ), VIRTUAL

**CreateHtml** Writes the HTML code representing the client area and its children.

*html object* The label of the HtmlClass object that writes the HTML code.

The **CreateHtml** method writes the HTML code representing the client area and its children.

CreateHtml is a VIRTUAL method so that other base class methods can directly call the CreateHtml virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The LayoutHtmlClass.CreateHtml method calls the WebAreaClass.CreateHtml method.

The CreateHtml method calls the WebWindowClass.CreateChildHtml method to write HTML for its child controls.

The CreateHtml method creates a special “continue” control for splash screens. See WebWindowClass.IsSplash.

Example:

```
WebToolBarClass.CreateHtml PROCEDURE(*HtmlClass Target)
```

```
CODE
SELF.PushFont(Target)
SELF.OwnerWindow.CreateChildHtml(Target, FEQ:ToolBar)
SELF.PopFont(Target)
```

See Also: LayoutHtmlClass.CreateHtml, WebWindowClass.CreateChildHtml

## GetBackgroundColor (return background color)

**GetBackgroundColor( [default color] ), LONG, VIRTUAL**

### GetBackgroundColor

Returns the background color of the Web page client area.

*default color* An integer variable, constant, EQUATE, or expression containing the color to return if there is no background color. If omitted, *default color* defaults to Color:None.

The **GetBackgroundColor** method returns the background color of the Web page client area.

GetBackgroundColor is a VIRTUAL method so that other base class methods can directly call the GetBackgroundColor virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The GetBackgroundColor method returns the COLOR attribute of the WINDOW control, if there is one. Otherwise, it returns the background color inherited from the Web page. EQUATES for the *default color* parameter are declared in \LIBSRC\EQUATES.CLW.

The CreateColorParameters method calls GetBackgroundColor.

Return Data Type: LONG

Example:

```
MyWebControlClass.CreateColorParameters PROCEDURE(*HtmlClass Target, BYTE AutoSpotLink)
ForeColor          LONG,AUTO
BackColor          LONG,AUTO
CODE
GETFONT(SELF.Feq,, ForeColor)
BackColor = SELF.GetBackgroundColor()
IF (ForeColor <> 0) AND (ForeColor <> COLOR:None)
    Target.WriteAppletParameter('ForeColor', IC:RGB(ForeColor))
END
IF (BackColor <> COLOR:None)
    Target.WriteAppletParameter('BackColor', IC:RGB(BackColor))
END
```

See Also: CreateColorParameters

## GetPosition (get control coordinates)

**GetPosition**( *x*, *y*, *width*, *height* ), **VIRTUAL**

<b>GetPosition</b>	Returns the client area's Web page coordinates.
<i>x</i>	A numeric variable to receive the control's horizontal position.
<i>y</i>	A numeric variable to receive the control's vertical position.
<i>width</i>	A numeric variable to receive the control's width.
<i>height</i>	A numeric variable to receive the control's height.

The **GetPosition** method returns the client area's Web page coordinates. The `LayoutHtmlClass` uses this information to position the client area on the Web page.

`GetPosition` is a **VIRTUAL** method so that other base class methods can directly call the `GetPosition` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Example:

```
LayoutHtmlClass.Insert PROCEDURE(*HtmlItemClass NewItem)

CurCell      &LayoutCellClass,AUTO
dx            SIGNED,AUTO
dy            SIGNED,AUTO
x             SIGNED,AUTO
y             SIGNED,AUTO
Xpos         SIGNED,AUTO
Ypos         SIGNED,AUTO
IsNew        BYTE,AUTO
CODE
NewItem.GetPosition(x, y, dx, dy)
Ypos = SELF.RangeY.AddPoint(y, dy, IsNew, SELF.SnapY)
IF (IsNew) THEN SELF.AddRow(Ypos).
Xpos = SELF.RangeX.AddPoint(x, dx, IsNew, SELF.SnapX)
IF (IsNew) THEN SELF.AddColumn(Xpos).
CurCell &= SELF.SetCell(Xpos, Ypos)
CurCell.Contents.Item &= NewItem
ADD(CurCell.Contents)
IF (dx > CurCell.dx)
    CurCell.dx = dx;
END
IF (dy > CurCell.dy)
    CurCell.dy = dy;
END
```



# ***WEB REPORT CLASS***

<b>Overview</b>	<b>443</b>
WebReportClass Concepts .....	443
Relationship to Other Internet Builder Classes .....	443
Internet Connect Template Implementation .....	443
Source Files .....	444
<b>WebReportClass Properties</b>	<b>445</b>
Html (HtmlClass object).....	445
Q (report image files to preview).....	445
NumPages (report page count) .....	446
Server (WebServerClass object) .....	446
<b>WebReportClass Methods</b>	<b>447</b>
Init (initialize the WebReportClass object) .....	447
Kill (shut down the WebReportClass object) .....	447
Preview (send report pages to client browser).....	448
SetNumPages (set maximum report page count) .....	448



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### WebReportClass Concepts

---

The WebReportClass object declares a simple report preview WINDOW. The window is designed to display the Windows metafiles generated by the Clarion PREVIEW statement (see the *Language Reference* for more information).

The WebReportClass uses the other IBC Library objects to “translate” this window, including the report preview images, to HTML, and to send the generated HTML to the Client browser for viewing.

### Relationship to Other Internet Builder Classes

---

The WebReportClass uses the WebServerClass, the WebWindowClass, the WebControlClass, and the HtmlClass. Therefore, if your program uses the WebReportClass, it also needs these other classes. However, the WebReportClass object instantiates most of these other objects as needed when you include the WebReportClass header file (ICREPORT.INC) in your program.

The WebReportClass does not instantiate its own WebServerClass or HtmlClass objects.

### Internet Connect Template Implementation

---

The Internet Connect Templates generate standard report preview code (the ReportPreview PROCEDURE) into the *appna\_SF.CLW* file. All Clarion template generated reports use this ReportPreview procedure to preview reports.

The ReportPreview procedure instantiates a WebReportClass object called HtmlPreview. If the Server application was launched by the Application Broker, then the ReportPreview procedure initializes, runs, and shuts down the HtmlPreview object instead of running the Windows report preview code.

The HtmlPreview object uses the global template incarnations of the WebServerClass (WebServer) and the HtmlClass (HtmlManager).

## Source Files

---

The WebReportClass source code is installed by default to the \LIBSRC folder. The source files and their components are as follows:

ICREPORT.INC	WebReportClass class declarations
ICREPORT.CLW	WebReportClass method definitions
ICREPORT.TRN	WebReportClass translation strings

## WebReportClass Properties

The WebReportClass contains the properties listed below.

### Html (HtmlClass object)

---

Html	&HtmlClass
	The <b>Html</b> property is a reference to the HtmlClass object that generates the HTML code that represents the report. The WebReportClass uses this property to generate HTML code.
Implementation:	The Init method sets the value of the Html property—typically to reference a global instance of the HtmlClass.
See Also:	Init

### Q (report image files to preview)

---

Q	&QUEUE, PROTECTED
	The <b>Q</b> property is a reference to a structure containing information about the report image files to preview. The WebReportClass uses this property to reference the files.
	This property is <b>PROTECTED</b> , therefore, it can only be referenced by a WebReportClass method, or a method in a class derived from WebReportClass.
Implementation:	The Init method sets the value of the Q property.
	The Q property is a reference to a <b>QUEUE</b> containing the pathnames, less the .wmf file extension, of the Windows metafiles to preview. The metafiles are typically generated by the <b>PREVIEW</b> statement (see the <i>Language Reference</i> for more information), one for each report page.
See Also:	Init

## NumPages (report page count)

---

### NumPages      SIGNED, PROTECTED

The **NumPages** property contains the number of report pages (the number of metafiles) to review. The `WebReportClass` object uses this property to show the end user how many report pages are available, and to control page scrolling.

This property is **PROTECTED**, therefore, it can only be referenced by a `WebReportClass` method, or a method in a class derived from `WebReportClass`.

Implementation:      The `SetNumPages` method sets the value of the `NumPages` property.

See Also:              `NumPages`

## Server (WebServerClass object)

---

### Server    &WebServerClass

The **Server** property is a reference to the `WebServerClass` object that represents the Server application for the `WebReportClass` object. The `WebReportClass` relies on this property to supply information about the Client browser, to supply appropriate pathnames, to handle events originating from the Client browser, and to interact with the `BrokerClass` object as needed.

Implementation:      The `Init` method sets the value of the `Server` property—typically to reference a global instance of the `WebServerClass`.

See Also:              `Init`

## WebReportClass Methods

The WebReportClass contains the methods listed below.

### Init (initialize the WebReportClass object)

---

**Init**( *server, html, previewQ* )

<b>Init</b>	Initializes the WebReportClass object.
<i>server</i>	The label of the WebServerClass object that represents the Sever (your Web-enabled application).
<i>html</i>	The label of the HtmlClass object that writes the HTML code representing the report pages.
<i>previewQ</i>	The label of a QUEUE or a field in a QUEUE that contains the names of the Windows metafiles for which to generate Web pages.

The **Init** method initializes the WebReportClass object.

Implementation: The Init method sets the initial values of the WebReportClass properties.

See Also: Html, Q, Server

### Kill (shut down the WebReportClass object)

---

**Kill, VIRTUAL**

The **Kill** method is only a virtual placeholder method to free any memory allocated during the life of the object and perform any other required termination code.

Kill is a VIRTUAL method so that other base class methods can directly call the Kill virtual method in a derived class. This lets you easily implement your own custom version of this method.

## Preview (send report pages to client browser)

---

### Preview, VIRTUAL

The **Preview** method generates and transmits the HTML representing the report page requested by the Client browser.

Preview is a VIRTUAL method so that other base class methods can directly call the Preview virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

The Preview class uses the Internet Connect window handling technology to display the report to the end user as a series of print preview windows within the Client browser.

The Preview method declares a basic report preview WINDOW, then uses the WebWindowClass to manage an iterative report preview session with the Client browser. That is, the Preview method sends HTML representing the initial print preview window, containing the first report page, to the Client browser. Then it accepts and processes any browser requests for additional report pages, by refreshing the print preview window and sending HTML to represent the refreshed window to the Client browser.

With the Preview method, the end user can only manipulate and print the report as a series of individual pages.

## SetNumPages (set maximum report page count)

---

### SetNumPages( [*number*] ), VIRTUAL

**SetNumPages** Sets the maximum previewable page count.

*number* The maximum number of pages to preview. If omitted, *number* defaults to zero (0).

The **SetNumPages** method sets the maximum previewable page count.

SetNumPages is a VIRTUAL method so that other base class methods can directly call the SetNumPages virtual method in a derived class. This lets you easily implement your own custom version of this method.

# ***JSL MANAGER CLASS***

<b>Overview</b>	<b>451</b>
JSLManagerClass Concepts .....	451
Relationship to Other Internet Builder Classes .....	452
Internet Connect Template Implementation .....	452
Source Files .....	452
<b>JSLManagerClass Properties</b>	<b>453</b>
Client (WebClientManagerClass object) .....	453
Files (WebFilesClass object) .....	453
Security (secure or public communication) .....	454
Target (TextOutputClass object) .....	454
<b>JSLManagerClass Methods</b>	<b>455</b>
Functional Organization—Expected Use .....	455
AddQueueEntry (add a Java list row) .....	456
BeginUpdate (write JSL data delimiters) .....	457
CloseChannel (close output channel and notify Client) .....	458
Init (initialize the JSLManagerClass object) .....	459
Kill (shut down the JSLManagerClass object) .....	459
OpenChannel (open channel for JSL data) .....	460
RemoveAllQueueEntries (delete all Java list rows) .....	462
RemoveQueueEntries (delete Java list rows) .....	463
ScrollQueueDown (scroll Java list down) .....	464
ScrollQueueUp (scroll Java list up) .....	465
SelectControl (set Java control to update) .....	466
SetAttribute (set Java control attribute—string) .....	467
SetAttributeFilename (set Java control attribute—pathname) .....	468
SetAttributeLong (set Java control attribute—number) .....	469
SetChecked (set Java check box status) .....	470
SetIconAttribute (display icon in Java list) .....	471
SetListChoice (highlight HTML list row) .....	472
SetQueueEntry (replace a Java list row) .....	473
SetValue (set Java control contents) .....	474



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

## JSLManagerClass Concepts

---

### Java Support Library (JSL)

TopSpeed provides the Java Support Library as part of the Internet Connect product. This library of Java objects resides on the Client computer. These Java objects must reside on the Client computer because the HTML code generated by your Web-enabled application relies on them. All Clarion Web-enabled applications share this single library.

The Java Support Library is very small and is automatically downloaded to the client computer as needed. The Client's browser and its caching options determine how the library is downloaded.

Some browsers download the entire library the first time the Client runs a Clarion Web-enabled application. Other browsers download individual library components as they are needed while running the Web-enabled application. In either case, the browsers generally cache the library (store it locally), and do not download it again unless the library is updated.

Many browsers let the end user set caching options which may affect Java Support Library downloads. For example, some browsers support a no caching option (not recommended), and many browsers also let you specify how often to check for newer versions of cached items—from never, to once-per-session, to every access.

### Efficient Partial Page Updates

The JSLManagerClass generates protocol and data for the Java Support Library objects on the Client computer. The JSLManagerClass provides methods that dynamically update some of the contents of an HTML page without refreshing the entire page in the browser. In other words, it does very efficient partial page updates over the internet. The JSLManagerClass accomplishes partial page updates by sending data (JSL data) only to the Java applets executing on the Client. The generated HTML page references the Java applets so they can refresh the browser's display without redrawing the entire page.

## Relationship to Other Internet Builder Classes

---

The `WebClientManagerClass` creates and manages a single `JSLManagerClass` object. The `WebControlClass` then uses the `JSLManagerClass` to do the partial updates.

## Internet Connect Template Implementation

---

The `WebClientManagerClass` creates and manages a single `JSLManagerClass` object. Therefore the template generated code does not directly reference the `JSLManagerClass` object.

## Source Files

---

The `JSLManagerClass` source code is installed by default to the `\LIBSRC` folder. The specific class declarations reside in the following files and their method definitions reside in the corresponding `.CLW` files.

`ICJSL.INC`

`JSLManagerClass`

## ***JSLManagerClass Properties***

The JSLManagerClass contains the properties listed below.

### **Client (WebClientManagerClass object)**

---

#### **Client    &WebClientManagerInterface, PROTECTED**

The **Client** property is a reference to the WebClientManagerClass object that represents the Client computer. The JSLManagerClass uses the Client property to notify the WebClientManagerClass object when JSL data is ready for transmittal to the Client computer.

This property is PROTECTED, therefore, it can only be referenced by a JSLManagerClass method, or a method in a class derived from JSLManagerClass.

Implementation:        The Init method sets the value of the Client property.

See Also:                Init

### **Files (WebFilesClass object)**

---

#### **Files     &WebFilesClass, PROTECTED**

The **Files** property is a reference to the WebFilesClass object that manages directories, pathnames, temporary files, aliases, etc for the JSLManagerClass object. The JSLManagerClass uses the Files property to provide appropriate pathnames for its files.

This property is PROTECTED, therefore, it can only be referenced by a JSLManagerClass method, or a method in a class derived from JSLManagerClass.

Implementation:        The OpenChannel method sets the value of the Files property.

See Also:                OpenChannel

## Security (secure or public communication)

---

### Security      SIGNED

The **Security** property contains a value indicating whether JSL data output by the JSLManagerClass are transmitted on a public channel or a secure channel.

There are significant performance penalties associated with secure communications. Therefore, we recommend using secure communications only when circumstances demand it.

Implementation:

The OpenChannel method sets the value of the Security property. EQUATE values for the Security property are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

See Also:

OpenChannel

## Target (TextOutputClass object)

---

### Target    &TextOutputClass, PROTECTED

The **Target** property is a reference to the TextOutputClass object that the JSLManagerClass uses to write the JSL data used to update the Java controls/applets. See *Text Output Class* for more information.

This property is PROTECTED, therefore, it can only be referenced by a JSLManagerClass method, or a method in a class derived from JSLManagerClass.

Implementation:

The Init method instantiates a NEW TextOuputClass object for the Target property.

See Also:

Init

## ***JSLManagerClass Methods***

The JSLManagerClass contains the methods listed below.

### **Functional Organization—Expected Use**

---

As an aid to understanding the JSLManagerClass, it is useful to organize the various JSLManagerClass methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### **Primary Interface Methods**

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize the JSLManagerClass object
OpenChannel	open an output channel
CloseChannel	finish data output and notify Client
Kill	shut down the JSLManagerClass object

##### **Mainstream Use:**

##### **Occasional Use:**

BeginUpdate	write JSL data delimiters
AddQueueEntry	add a Java list row
RemoveAllQueueEntries	delete all Java list rows
RemoveQueueEntries	delete Java list rows
ScrollQueueDown	scroll Java list down
ScrollQueueUp	scroll Java list up
SelectControl	set Java control to update
SetAttribute	set Java control attribute—string
SetAttributeLong	set Java control attribute—number
SetAttributeFilename	set Java control attribute—pathname
SetChecked	set Java check box status
SetIconAttribute	display icon in Java list
SetListChoice	highlight Java list row
SetQueueEntry	replace a Java list row
SetValue	set Java control contents

#### **Virtual Methods**

The JSLManagerClass has no virtual methods.

## AddQueueEntry (add a Java list row)

**AddQueueEntry**( *control*, *position* [,*format*] )

<b>AddQueueEntry</b>	Adds a row to a Java list.
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Clarion control that is the data source for the Java list. This is the Clarion field equate number.
<i>position</i>	An integer constant, variable, EQUATE, or expression containing the row number from the <i>control</i> to add.
<i>format</i>	An integer constant, variable, EQUATE, or expression containing ... If omitted, <i>format</i> defaults to zero (0).

The **AddQueueEntry** method adds a new row to the Java list.

The *control* parameter identifies the Clarion control that is the data source for the new Java list row. The *position* parameter identifies the specific row to add to the Java list.

The **SelectControl** method sets the target Java list to update.

Example:

```
WebJavaListClass.CreateJsldata PROCEDURE(*JsIManagerClass Target)
!procedure data
  CODE
  !procedure code
  NumActions = RECORDS(SELF.QueueActionQ)
  LOOP CurAction = 1 TO NumActions
    GET(SELF.QueueActionQ, CurAction)
    NumActionItems = SELF.QueueActionQ.NumItems
    CASE (SELF.QueueActionQ.Action)
    OF ACTION:Insert
      LOOP CurRecord = 1 TO NumActionItems
        Target.AddQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
      END
    OF ACTION:Replace
      LOOP CurRecord = 1 TO NumActionItems
        Target.SetQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
      END
    OF ACTION:Delete
      Target.RemoveQueueEntries(SELF.QueueActionQ.Offset, NumActionItems)
    OF ACTION:DeleteAll
      Target.RemoveAllQueueEntries
    END
  END
!procedure code
```

See Also: **SelectControl**

## BeginUpdate (write JSL data delimiters)

---

### BeginUpdate( *applet* )

**BeginUpdate** Writes JSL data delimiters to indicate a new *control* is being addressed.

*applet* An integer constant, variable, EQUATE, or expression containing the control/applet number to update as set by the SelectControl method; it is *not* the Clarion field equate number.

The **BeginUpdate** method writes appropriate Java Support Library protocol delimiters to identify which *applet* is being addressed in the generated JSL data. The BeginUpdate method determines whether the generated data applies to a new control/applet, and writes any necessary delimiters to “end” one control and “begin” another.

Example:

```
Js1ManagerClass.SetAttribute PROCEDURE(BYTE Which)
CODE
SELF.BeginUpdate(SELF.NextId)
SELF.Target.WriteLine(CHR(Which))
```

See Also:            **SelectControl**

## CloseChannel (close output channel and notify Client)

---

### CloseChannel

The **CloseChannel** method writes end-of-control and end-of-code delimiters, then closes the output channel. Finally, the CloseChannel method notifies the Client (property) that the JSL data is complete and ready for further processing.

#### Implementation:

When the CloseChannel method notifies the WebClientManagerClass object that the JSL data is complete, the WebClientManagerClass object then notifies the BrokerClass object. The BrokerClass object supplies an appropriate Http header for the JSL data, then notifies the Application Broker that the JSL data is ready for transmittal to the Client computer.

The WebWindowClass.TakeCreatePage method opens and closes the channel as needed to generate JSL data for its window.

#### Example:

```
MyWebWindowClass.TakeCreatePage  PROCEDURE
Client      &WebClientManagerClass
CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
!generate whole HTML page
ELSE
Client.Jsl.OpenChannel(SELF.GetTargetSecurity(),SELF.Files)
SELF.CreateJslData(Client.Jsl)
Client.Jsl.CloseChannel
END
SELF.Server.TakePageSent()
```

#### See Also:

Client, WebWindowClass.TakeCreatePage

## Init (initialize the JSLManagerClass object)

---

### Init( *client* )

**Init**                      Initializes the JSLManagerClass object.  
*client*                      The label of the WebClientManagerClass object that has requested data.

The **Init** method initializes the JSLManagerClass object.

Implementation:            The Init method sets the Client property to reference the *client* and instantiates a NEW TextOutputClass object for the Target property.

Example:

```
WebClientManagerClass.Init PROCEDURE |
    (*WebDataSinkClass Broker,*BrowserManagerClass Browser)
CODE
SELF.Js1 &= NEW Js1ManagerClass
SELF.Js1.Init(SELF)
SELF.Broker &= Broker
SELF.Browser &= Browser
```

See Also:                    Client, Target

## Kill (shut down the JSLManagerClass object)

---

### Kill

The **Kill** method frees any memory allocated during the life of the object and does any other required termination code.

Implementation:            The Kill method DISPOSEs the Target property's TextOutputClass object .

Example:

```
WebClientManagerClass.Kill PROCEDURE
CODE
IF (NOT SELF.Js1 &= NULL)
    SELF.Js1.Kill
DISPOSE(SELF.Js1)
END
```

## OpenChannel (open channel for JSL data)

### OpenChannel( *security*, *files object* )

<b>OpenChannel</b>	Opens a channel to write data for the Java Support Library.
<i>security</i>	An integer constant, variable, EQUATE, or expression indicating whether to use a secure channel or a public channel.
<i>files object</i>	The label of the WebFilesClass object that manages directories, pathnames, etc. for this JSLManagerClass object.

The **OpenChannel** method opens a channel to write data for the Java Support Library (JSL data). The JSL data allows very efficient partial page updates to Server generated Web pages. The JSL data is applied by the Java Support Library on the Client computer.

There are significant performance penalties associated with secure communications. Therefore, we recommend using secure channels only when circumstances demand them.

#### Implementation:

The OpenChannel method sets the value of the Files property based on the value of the *files object* parameter. Other JSLManagerClass methods use the Files property to provide appropriate pathnames wherever they are needed.

The OpenChannel method sets the value of the Security property based on the value of the *security* parameter. EQUATEs for the *security* parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

The WebWindowClass.TakeCreatePage method opens and closes a “channel” as needed to generate JSL data for its window.

Example:

```
MyWebWindowClass.TakeCreatePage PROCEDURE
Client      &WebClientManagerClass
CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
    !generate whole HTML page
ELSE
    Client.Js1.OpenChannel(SELF.GetTargetSecurity(),SELF.Files)
    SELF.CreateJs1Data(Client.Js1)
    Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()
```

See Also: [Security](#), [Files](#), [WebWindowClass.TakeCreatePage](#)

## RemoveAllQueueEntries (delete all Java list rows)

---

### RemoveAllQueueEntries

The **RemoveAllQueueEntries** method deletes all the rows from the Java list control set by the **SelectControl** method.

Example:

```
WebJavaListClass.CreateJsldata PROCEDURE(*JsIManagerClass Target)
!procedure data
CODE
!procedure code
NumActions = RECORDS(SELF.QueueActionQ)
LOOP CurAction = 1 TO NumActions
  GET(SELF.QueueActionQ, CurAction)
  NumActionItems = SELF.QueueActionQ.NumItems
  CASE (SELF.QueueActionQ.Action)
  OF ACTION:Insert
    LOOP CurRecord = 1 TO NumActionItems
      Target.AddQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
    END
  OF ACTION:Replace
    LOOP CurRecord = 1 TO NumActionItems
      Target.SetQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
    END
  OF ACTION>Delete
    Target.RemoveQueueEntries(SELF.QueueActionQ.Offset, NumActionItems)
  OF ACTION>DeleteAll
    Target.RemoveAllQueueEntries
  END
END
!procedure code
```

See Also: [SelectControl](#)

## RemoveQueueEntries (delete Java list rows)

**RemoveQueueEntries**( *position*, *number* )

### RemoveQueueEntries

Deletes *number* rows starting from row *position* from a Java list control.

*position* An integer constant, variable, EQUATE, or expression containing the number of the first row to delete.

*number* An integer constant, variable, EQUATE, or expression containing the number of rows to delete.

The **RemoveQueueEntries** method deletes *number* rows starting from row *position* in the Java list control set by the **SelectControl** method.

Example:

```
WebJavaListClass.CreateJsldata PROCEDURE(*JsIManagerClass Target)
!procedure data
  CODE
  !procedure code
  NumActions = RECORDS(SELF.QueueActionQ)
  LOOP CurAction = 1 TO NumActions
    GET(SELF.QueueActionQ, CurAction)
    NumActionItems = SELF.QueueActionQ.NumItems
    CASE (SELF.QueueActionQ.Action)
    OF ACTION:Insert
      LOOP CurRecord = 1 TO NumActionItems
        Target.AddQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
      END
    OF ACTION:Replace
      LOOP CurRecord = 1 TO NumActionItems
        Target.SetQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
      END
    OF ACTION>Delete
      Target.RemoveQueueEntries(SELF.QueueActionQ.Offset, NumActionItems)
    OF ACTION>DeleteAll
      Target.RemoveAllQueueEntries
    END
  END
!procedure code
```

See Also: [SelectControl](#)

## ScrollQueueDown (scroll Java list down)

**ScrollQueueDown**( *control*, *number* [,*format*] )

**ScrollQueueDown** Scrolls a Java list downward *number* rows.

*control* An integer constant, variable, EQUATE, or expression containing the control number of the Clarion control that is the data source for the Java list. This is the Clarion field equate number.

*number* An integer constant, variable, EQUATE, or expression containing the number of rows to scroll from the current position.

*format* An integer constant, variable, EQUATE, or expression containing ... If omitted, *format* defaults to zero (0).

The **ScrollQueueDown** method scrolls a Java list downward *number* rows. The *control* parameter identifies the Clarion control that is the data source for the Java list.

The SelectControl method sets the list to scroll.

Example:

```
WebJavaListClass.CreateJsldata PROCEDURE(*JsIManagerClass Target)
!procedure data
CODE
!procedure code
NumActions = RECORDS(SELF.QueueActionQ)
LOOP CurAction = 1 TO NumActions
GET(SELF.QueueActionQ, CurAction)
NumActionItems = SELF.QueueActionQ.NumItems
CASE (SELF.QueueActionQ.Action)
OF ACTION:ScrollUp
LOOP CurRecord = 1 TO NumActionItems
Target.ScrollQueueUp(CurFeq, NumRecords - NumActionItems + CurRecord, SELF.Format)
END
OF ACTION:ScrollDown
LOOP CurRecord = 1 TO NumActionItems
Target.ScrollQueueDown(CurFeq, NumActionItems - (CurRecord - 1), SELF.Format)
END
!other actions
END
END
!procedure code
```

See Also: [SelectControl](#)

## ScrollQueueUp (scroll Java list up)

**ScrollQueueUp**( *control*, *number* [,*format*] )

<b>ScrollQueueUp</b>	Scrolls a Java list upward <i>number</i> rows.
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Clarion control that is the data source for the Java list. This is the Clarion field equate number.
<i>number</i>	An integer constant, variable, EQUATE, or expression containing the number of rows to scroll from the current position.
<i>format</i>	An integer constant, variable, EQUATE, or expression containing ... If omitted, <i>format</i> defaults to zero (0).

The **ScrollQueueUp** method scrolls a Java list upward *number* rows. The *control* parameter identifies the Clarion control that is the data source for the Java list.

The **SelectControl** method sets the list to scroll.

Example:

```
WebJavaListClass.CreateJsldata PROCEDURE(*Js1ManagerClass Target)
!procedure data
  CODE
  !procedure code
  NumActions = RECORDS(SELF.QueueActionQ)
  LOOP CurAction = 1 TO NumActions
    GET(SELF.QueueActionQ, CurAction)
    NumActionItems = SELF.QueueActionQ.NumItems
    CASE (SELF.QueueActionQ.Action)
    OF ACTION:ScrollUp
      LOOP CurRecord = 1 TO NumActionItems
        Target.ScrollQueueUp(CurFeq,NumRecords-NumActionItems+CurRecord,SELF.Format)
      END
    OF ACTION:ScrollDown
      LOOP CurRecord = 1 TO NumActionItems
        Target.ScrollQueueDown(CurFeq,NumActionItems-(CurRecord-1),SELF.Format)
      END
    !other actions
  END
END
!procedure code
```

See Also: **SelectControl**

## SelectControl (set Java control to update)

---

### SelectControl( *control* )

**SelectControl** Sets the current Java control/applet for other JSLManagerClass methods.

*control* An integer constant, variable, EQUATE, or expression containing the control number of the corresponding Clarion control. This is the Clarion field equate number.

The **SelectControl** method sets the current Java control/applet for other JSLManagerClass methods. Other JSLManagerClass methods generate protocol and data to update the specified control's attributes, contents, etc.

Example:

```
WebJavaImageClass.CreateJs1Data PROCEDURE(*Js1ManagerClass Target)
Filename CSTRING(FILE:MaxFileName),AUTO
CODE
Filename = SELF.Feq{PROP:tempimage}
Target.SelectControl(SELF.Feq)
IF (SELF.Filename <> Filename)
    Target.SetAttributeFilename(JSL:Picture, Filename)
    SELF.Filename = Filename
END
```

## SetAttribute (set Java control attribute—string)

**SetAttribute**( *attribute* [, *value*] )

<b>SetAttribute</b>	Sets a Java control attribute.
<i>attribute</i>	An integer constant, variable, EQUATE, or expression indicating which attribute to set.
<i>value</i>	A string constant, variable, EQUATE, or expression containing the value of the <i>attribute</i> . If omitted, the <i>attribute</i> is Boolean and is toggled from its current setting to its opposite setting.

The **SetAttribute** method sets a Java control's attribute. The **SelectControl** method determines the control whose *attribute* is set.

Implementation:

EQUATEs for the *attribute* parameter are declared in ICJSL.INC as follows:

```

ITEMIZE,PRE(JSL)
ListRecords      EQUATE(VAL('R'))    !use SetAttributeLong
ListFields       EQUATE(VAL('F'))    !use SetAttributeLong
ListSelection    EQUATE(VAL('S'))    !use SetAttributeLong
ListThumb       EQUATE(VAL('M'))    !use SetAttributeLong
ListHScroll     EQUATE(VAL('H'))    !use SetAttribute
ListVScroll     EQUATE(VAL('V'))    !use SetAttribute
ListFormat      EQUATE(VAL('A'))    !use SetAttribute
ListIcon        EQUATE(VAL('I'))    !use SetIconAttribute
Picture         EQUATE(VAL('P'))    !use SetAttributeFilename
Disable         EQUATE(VAL('D'))    !use SetAttribute
Enable         EQUATE(VAL('E'))    !use SetAttribute
StringText     EQUATE(VAL('T'))    !use SetAttribute
END

```

Example:

```

WebJavaStringClass.CreateJs1Data      PROCEDURE(*Js1ManagerClass Target)
CurText                               ANY
CODE
CurText = SELF.GetText()
IF (CurText <> SELF.LastText)
    Target.SelectControl(SELF.Feq)
    Target.SetAttribute(Js1:StringText, CurText)
    SELF.LastText = CurText
END

```

See Also:

**SelectControl**

## SetAttributeFilename (set Java control attribute—pathname)

**SetAttributeFilename**( *attribute*, *filename* )

### SetAttributeFilename

Sets a Java control attribute to contain a pathname.

*attribute* An integer constant, variable, EQUATE, or expression indicating which attribute to set.

*filename* A string constant, variable, EQUATE, or expression containing only the filename, with no path prepended.

The **SetAttributeFilename** method sets a Java control's *attribute* to contain a pathname. The pathname consists of the path concatenated to the *filename*. The Files property supplies the appropriate path value.

The SelectControl method sets the control whose *attribute* is set.

Implementation:

EQUATEs for the *attribute* parameter are declared in ICJSL.INC as follows:

```

ITEMIZE, PRE(JSL)
ListRecords    EQUATE(VAL('R'))    !use SetAttributeLong
ListFields     EQUATE(VAL('F'))    !use SetAttributeLong
ListSelection  EQUATE(VAL('S'))    !use SetAttributeLong
ListThumb      EQUATE(VAL('M'))    !use SetAttributeLong
ListHScroll    EQUATE(VAL('H'))    !use SetAttribute
ListVScroll    EQUATE(VAL('V'))    !use SetAttribute
ListFormat     EQUATE(VAL('A'))    !use SetAttribute
ListIcon       EQUATE(VAL('I'))    !use SetIconAttribute
Picture        EQUATE(VAL('P'))    !use SetAttributeFilename
Disable        EQUATE(VAL('D'))    !use SetAttribute
Enable         EQUATE(VAL('E'))    !use SetAttribute
StringText     EQUATE(VAL('T'))    !use SetAttribute
END

```

Example:

```

WebJavaImageClass.CreateJslData PROCEDURE(*JslManagerClass Target)
Filename CSTRING(FILE:MaxFileName),AUTO
CODE
Filename = SELF.Feq{PROP:tempimage}
Target.SelectControl(SELF.Feq)
IF (SELF.Filename <> Filename)
    Target.SetAttributeFilename(JSL:Picture, Filename)
    SELF.Filename = Filename
END

```

See Also:

Files, SelectControl

## SetAttributeLong (set Java control attribute—number)

**SetAttribute**( *attribute*, *value* )

<b>SetAttribute</b>	Sets a Java control attribute.
<i>attribute</i>	An integer constant, variable, EQUATE, or expression indicating which attribute to set.
<i>value</i>	An integer constant, variable, EQUATE, or expression containing the value of the <i>attribute</i> .

The **SetAttribute** method sets a Java control's attribute. The **SelectControl** method determines the control whose *attribute* is set.

Implementation: EQUATEs for the *attribute* parameter are declared in ICJSL.INC as follows:

```

ITEMIZE, PRE(JSL)
ListRecords      EQUATE(VAL('R'))    !use SetAttributeLong
ListFields       EQUATE(VAL('F'))    !use SetAttributeLong
ListSelection    EQUATE(VAL('S'))    !use SetAttributeLong
ListThumb        EQUATE(VAL('M'))    !use SetAttributeLong
ListHScroll      EQUATE(VAL('H'))    !use SetAttribute
ListVScroll      EQUATE(VAL('V'))    !use SetAttribute
ListFormat       EQUATE(VAL('A'))    !use SetAttribute
ListIcon         EQUATE(VAL('I'))    !use SetIconAttribute
Picture          EQUATE(VAL('P'))    !use SetAttributeFilename
Disable          EQUATE(VAL('D'))    !use SetAttribute
Enable           EQUATE(VAL('E'))    !use SetAttribute
StringText       EQUATE(VAL('T'))    !use SetAttribute
END

```

Example:

```

WebJavaListClass.CreateJslData      PROCEDURE(*JslManagerClass Target)
!procedure data
CODE
!procedure code
IF (CurFeq{PROP:imm})
    Target.SetAttributeLong(JSL:ListThumb, CurFeq{PROP:vscrollpos})
END
IF (SELF.LastHscroll <> CurFeq{PROP:hscroll})
    Target.SetAttributeLong(JSL:ListHScroll, CurFeq{PROP:hscroll})
END
IF (SELF.LastVscroll <> CurFeq{PROP:vscroll})
    Target.SetAttributeLong(JSL:ListVScroll, CurFeq{PROP:hscroll})
END
!procedure code

```

See Also: **SelectControl**

## SetChecked (set Java check box status)

---

### SetChecked( *control*, *status* )

<b>SetChecked</b>	Sets the <i>status</i> of a Java check box.
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Clarion control that corresponds to the Java check box. This is the Clarion field equate number.
<i>status</i>	An integer constant, variable, EQUATE, or expression indicating whether the box is checked. A value of one (1) indicates the box is checked; a value of zero (0) indicates the box is not checked.

The **SetChecked** method sets the the *status* of a Java check box. The *control* parameter determines which Java check box is set. The *status* parameter determines whether or not the box is checked.

Example:

```
WebHtmlCheckClass.CreateJs1Data      PROCEDURE(*Js1ManagerClass Target)

CODE
IF (SELF.GetUserChanged())
    Target.SetChecked(SELF.Feq, SELF.Feq{PROP:checked})
    SELF.UpdateCopyUse
END
```

## SetIconAttribute (display icon in Java list)

**SetIconAttribute**( *row*, *filename* )

**SetIconAttribute** Sets a Java list icon attribute to contain a pathname.

*row* An integer constant, variable, EQUATE, or expression indicating which row displays the specified icon. This is the Clarion row number.

*filename* A string constant, variable, EQUATE, or expression containing the icon filename, with no path prepended.

The **SetIconAttribute** method sets a Java list's icon attribute to contain a pathname. The pathname consists of the path concatenated to the *filename*. The Files property supplies the appropriate path value.

The SelectControl method sets the list whose icon attribute is set.

Example:

```
WebJavaListClass.CreateJsldata      PROCEDURE(*JsManagerClass Target)
!procedure data
  CODE
!procedure code
  IF (SELF.SendIcons)
    CurIndex = 0
    Skipped = 0
    LOOP WHILE (Skipped < 2)
      NextIconName = SELF.OwnerWindow.Files.LoadImage(CurFreq{PROP:iconlist, CurIndex})
      IF (NextIconName)
        Target.SetIconAttribute(CurIndex, NextIconName)
        Skipped = 0
      ELSE
        Skipped += 1
      END
      CurIndex += 1
    END
    SELF.SendIcons = FALSE
  END
```

See Also: [Files](#), [SelectControl](#)

## SetListChoice (highlight HTML list row)

---

**SetListChoice**( *control*, *row* )

<b>SetListChoice</b>	Set an HTML list's selected attribute.
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Clarion control that is the data source for the Java list. This is the Clarion field equate number.
<i>row</i>	An integer constant, variable, EQUATE, or expression indicating which row is selected or highlighted. This is the Clarion row number.

The **SetListChoice** method sets an HTML list's selected attribute. The *control* parameter determines which list control is updated. The *row* parameter determines which row is selected or highlighted.

Example:

```
WebHtmlListClass.CreateJs1Data      PROCEDURE(*Js1ManagerClass Target)

CODE
IF (SELF.GetChoiceChanged())
    Target.SetListChoice(SELF.Feq, CHOICE(SELF.Feq))
    SELF.UpdateCopyChoice
END
```

## SetQueueEntry (replace a Java list row)

**SetQueueEntry**( *control*, *row* [, *format*] )

<b>SetQueueEntry</b>	Replace a Java list row.
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Clarion control that is the data source for the Java list. This is the Clarion field equate number.
<i>row</i>	An integer constant, variable, EQUATE, or expression indicating the row to replace. This is the Clarion row number.
<i>format</i>	An integer constant, variable, EQUATE, or expression containing ... If omitted, <i>format</i> defaults to zero (0).

The **SetQueueEntry** method replaces a row in a Java list. The *control* parameter determines which list control is updated. The *row* parameter determines which row is replaced.

Example:

```

WebJavaListClass.CreateJsldata PROCEDURE(*JsIManagerClass Target)
!procedure data
  CODE
  !procedure code
  NumActions = RECORDS(SELF.QueueActionQ)
  LOOP CurAction = 1 TO NumActions
    GET(SELF.QueueActionQ, CurAction)
    NumActionItems = SELF.QueueActionQ.NumItems
    CASE (SELF.QueueActionQ.Action)
    OF ACTION:Insert
      LOOP CurRecord = 1 TO NumActionItems
        Target.AddQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
      END
    OF ACTION:Replace
      LOOP CurRecord = 1 TO NumActionItems
        Target.SetQueueEntry(CurFeq,SELF.QueueActionQ.Offset+(CurRecord-1),SELF.Format)
      END
    OF ACTION>Delete
      Target.RemoveQueueEntries(SELF.QueueActionQ.Offset, NumActionItems)
    OF ACTION>DeleteAll
      Target.RemoveAllQueueEntries
    END
  END
!procedure code

```

## SetValue (set Java control contents)

---

**SetValue**( *control*, *value* )

<b>SetValue</b>	Sets a Java control's contents.
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Clarion control that corresponds to the Java control. This is the Clarion field equate number.
<i>value</i>	A constant, variable, EQUATE, or expression containing the value to assign to the Java control.

The **SetValue** method sets a Java control's contents.

Example:

```
WebHtmlEntryClass.CreateJs1Data      PROCEDURE(*Js1ManagerClass Target)

CODE
IF (SELF.GetUseChanged())
    Target.SetValue(SELF.Feq, CONTENTS(SELF.Feq))
    SELF.UpdateCopyUse
END
```

# ***JSL EVENTS CLASS***

<b>Overview</b>	<b>477</b>
JsEventsClass Concepts .....	477
Relationship to Other Internet Builder Classes .....	478
Internet Connect Template Implementation .....	478
Source Files .....	478
<b>JsEventsClass Properties</b>	<b>479</b>
EventQ (equivalent Clarion and JSL events) .....	479
<b>JsEventsClass Methods</b>	<b>480</b>
Functional Organization—Expected Use .....	480
AddEvent (add a Clarion/JSL event pair) .....	481
Init (initialize the JsEventsClass object) .....	482
Kill (shut down JsEventsClass object) .....	483
GetEventNumber (return Clarion event number) .....	484
GetEventString (return event/action pairs) .....	485



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

## JslEventsClass Concepts

---

The JslEventsClass manages the mapping between Clarion events and Java Support Library events (JSL events). See *JSL Manager Class* for more information on the Java Support Library. This event mapping is primarily used when handling LISTS. That is, since HTML has no equivalent to a Clarion LIST and its associated events, Internet Connect relies on a Java list applet to represent lists and their associated events on a Web page. The JslEventsClass supports the following standard Clarion events:

```
EVENT:Accepted  
  
EVENT:Initialize  
EVENT:NewSelection  
EVENT:AlertKey  
EVENT:Locate  
  
EVENT:ScrollTop  
EVENT:ScrollBottom  
EVENT:PageUp  
EVENT:PageDown  
EVENT:ScrollUp  
EVENT:ScrollDown  
EVENT:ScrollDrag  
  
EVENT:Expanding  
EVENT:Contracting  
EVENT:Expanded  
EVENT:Contracted
```

Typically, your Web-enabled application instantiates a single global JSLEventsClass object.

### Java Support Library

TopSpeed provides the Java Support Library as part of the Internet Connect product. This library of Java objects must reside on the client computer because the HTML code your Web-enabled application generates relies on these objects. The Java Support Library is very small and is automatically downloaded to the client computer as needed by the Clarion Application Broker. (see *JSL Manager Class* for more information).

## Relationship to Other Internet Builder Classes

---

The SubmitItem class uses the JSLEventsClass to convert Java Support Library events to standard Clarion event EQUATES. The WebJavaListClass uses the JSLEventsClass to set the available events for the Java list applets that represent Clarion LISTS within the HTML code generated by your Web-enabled application.

## Internet Connect Template Implementation

---

The Internet Connect Templates generate code to instantiate a single global JSLEventsClass object for your application program. The JSLEventsClass object is called JavaEvents. The template code only references the JavaEvents object to declare it, initialize it, and shut it down. Other IBC Library objects reference the JavaEvents object as an EXTERNAL.

## Source Files

---

The JSLEventsClass source code is installed by default to the \LIBSRC folder. The specific class declarations reside in the following files and their method definitions reside in the corresponding .CLW files.

ICEVENTS.INC JSLEventsClass

## *JslEventsClass Properties*

The JslEventsClass contains only one property.

### **EventQ (equivalent Clarion and JSL events)**

---

#### **EventQ**

#### **&EventInfoQueue**

The **EventQ** property is a reference to a structure containing a list of Clarion event EQUATE numbers and their corresponding Java Support Library event codes. For example, a Clarion EVENT:PageDown maps to JSL event 'B', and a Clarion EVENT:Accepted maps to JSL event 'X'.

#### Implementation:

The Init method loads the EventQ property by making several calls to the AddEvent method.

The EventQ property is a reference to QUEUE declared in ICEVENTS.INC as follows:

```
EventInfoQueue  QUEUE,TYPE
EventNo         SIGNED
Letter         STRING(1)
END
```

#### See Also:

AddEvent, Init

## ***JSLEventsClass Methods***

The JSLEventsClass contains the methods listed below.

### **Functional Organization—Expected Use**

---

As an aid to understanding the JSLEventsClass, it is useful to organize its methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### **Primary Interface Methods**

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize the JSLEventsClass object
Kill	shut down JSLEventsClass object

##### **Mainstream Use:**

##### **Occasional Use:**

AddEvent	add an Clarion/JSL events pair
GetEventNumber	returnClarion event number
GetEventString	return Clarion-event/Web-action pairs

#### **Virtual Methods**

The JSLEventsClass has no virtual methods.

## AddEvent (add a Clarion/JSL event pair)

---

### AddEvent( *Clarion event, JSL event* )

<b>AddEvent</b>	Adds a Clarion/JSL event pair to the EventQ property.
<i>Clarion event</i>	An integer constant, variable, EQUATE, or expression containing the Clarion event number that is equivalent to the <i>JSL event</i> .
<i>JSL event</i>	A string constant, variable, EQUATE, or expression containing the JSL event value that is equivalent to the <i>Clarion event</i> .

The **AddEvent** method adds a Clarion event number and its corresponding JSL event code to the EventQ property.

Implementation: The AddEvent method adds items to the EventQ property. It sorts the EventQ by *JSL events*.

Example:

```
Js1EventsClass.Init PROCEDURE  
  
CODE  
SELF.EventQ &= NEW EventInfoQueue  
SELF.AddEvent(EVENT:ScrollBottom, 'B')  
SELF.AddEvent(EVENT:PageDown, 'C')  
SELF.AddEvent(EVENT:ScrollDown, 'D')  
SELF.AddEvent(EVENT:Expanding, 'E')  
SELF.AddEvent(EVENT:Initialize, 'I')  
SELF.AddEvent(EVENT:AlertKey, 'K')  
SELF.AddEvent(EVENT:Locate, 'L')  
SELF.AddEvent(EVENT:Contracting, 'O')  
SELF.AddEvent(EVENT:ScrollDrag, 'M')  
SELF.AddEvent(EVENT:NewSelection, 'N')  
SELF.AddEvent(EVENT:PageUp, 'R')  
SELF.AddEvent(EVENT:ScrollTop, 'T')  
SELF.AddEvent(EVENT:ScrollUp, 'U')  
SELF.AddEvent(EVENT:Accepted, 'X')  
SELF.AddEvent(EVENT:Expanded, 'Y')  
SELF.AddEvent(EVENT:Contracted, 'Z')
```

See Also: [EventQ](#)

## Init (initialize the JSLEventsClass object)

### Init

The **Init** method initializes the JSLEventsClass object.

Implementation: The Init method allocates a new EventQ property and loads it with the default event pairs with multiple calls to the AddEvent method.

Example:

```

Broker          BrokerClass
HtmlManager     HtmlClass
JavaEvents      JsLEventsClass      !declare JavaEvents object
WebServer       WebServerClass
WebFileManager  WebFilesClass

ICServerWin     WINDOW,AT(-100,-100,0,0)
                END

CODE
SetWebActiveFrame
WebFileManager.Init(1, '')
JavaEvents.Init      !initialize JavaEvents object
Broker.Init('TREE', WebFileManager)
HtmlManager.Init(WebFileManager)
WebServer.Init(Broker, '', 600, '', WebFileManager)
IF (WebServer.GetInternetEnabled())
  OPEN(ICServerWin)
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect
    WebMain
    BREAK
  END
END
END
WebServer.Kill
HtmlManager.Kill
Broker.Kill()
JavaEvents.Kill      !shut down JavaEvents object
WebFileManager.Kill

```

See Also: AddEvent, EventQ

## Kill (shut down JSLEventsClass object)

### Kill

The **Kill** method frees any memory allocated during the life of the object and does any other required termination code.

Implementation: The Init method fress the EventQ property.

Example:

```

Broker          BrokerClass
HtmlManager     HtmlClass
JavaEvents      JslEventsClass           !declare JavaEvents object
WebServer       WebServerClass
WebFilesManager WebFilesClass

ICServerWin     WINDOW,AT(-100,-100,0,0)
                END

CODE
SetWebActiveFrame
WebFilesManager.Init(1, '')
JavaEvents.Init           !initialize JavaEvents object
Broker.Init('TREE', WebFilesManager)
HtmlManager.Init(WebFilesManager)
WebServer.Init(Broker, '', 600, '', WebFilesManager)
IF (WebServer.GetInternetEnabled())
  OPEN(ICServerWin)
  ACCEPT
  IF (EVENT() = EVENT:OpenWindow)
    WebServer.Connect
    WebMain
  BREAK
END
END
WebServer.Kill
HtmlManager.Kill
Broker.Kill()
JavaEvents.Kill           !shut down JavaEvents object
WebFilesManager.Kill

```

## GetEventNumber (returnClarion event number)

### GetEventNumber( *JSL events* ), SIGNED

**GetEventNumber** Returns the Clarion event number corresponding to the specified *JSL event*.

*JSL event* A string constant, variable, EQUATE, or expression containing a JSL event code.

The **GetEventNumber** method returns the Clarion event number corresponding to the specified *JSL event*. The SubmitItem class uses GetEventNumber to convert JSL event codes to their equivalent Clarion event numbers.

Implementation: If there is no corresponding event in the EventQ property, GetEventNumber returns EVENT:Accepted.

Return Data Type: SIGNED

Example:

```
SubmitItemClass.Reset PROCEDURE(Arguments)
!procedure data
EventNo                SIGNED
CODE
!procedure code to set EventPos and EndPos
EventNo = EVENT:Accepted
IF (EventPos < EndPos)
    EventNo = JavaEvents.GetEventNumber(Arguments[EventPos])
END

SELF.Event = EventNo
ControlId = SUB(Arguments, StartPos, EventPos-StartPos)
SELF.Name = Arguments[(1):(AssignPos-1)]
SELF.Extra = SUB(Arguments, EventPos+1, AssignPos - (EventPos+1))
SELF.NewValue = SUB(Arguments, AssignPos+1, EndPos - (AssignPos+1))
!procedure code
```

See Also: EventQ, SubmitItemClass.Reset

## GetEventString (return event/action pairs)

**GetEventString( *event action queue*, *default action* ), **STRING****

**GetEventString** Returns vertical bar (|) delimited event/action pairs.

*event action queue* The label of a structure that associates Clarion event numbers with appropriate Web page actions for a specific control. Typically, this is the `WebJavaListClass.EventActionQ` property.

*default action* An integer constant, variable, EQUATE, or expression containing the default action.

The **GetEventString** method returns a vertical bar (|) delimited event/action pair for each event in the EventQ property. Typically, the *event action queue* contains the developer specified Web page actions for a single list control (see `WebJavaListClass.EventActionQ`).

Implementation:

The `WebJavaListClass` uses the return string as the *Events* parameter to the Java list applet when it generates HTML to represent a Clarion LIST.

The return string is of the form '*E9|E9*' where *E* is a JSL event code from the EventQ property and *9* is the corresponding action code. Standard Web page actions and their codes are represented by EQUATEs declared in `ICSTD.EQU` as follows:

```
Update:OnBrowser  EQUATE(0)
Update:Partial   EQUATE(1)
Update:Full      EQUATE(2)
Update:Refresh   EQUATE(3)
```

If a JSL event has no corresponding action in the *event action queue*, then the JSL event is paired with the *default action*, typically `Update:OnBrowser`.

The *event action queue* parameter must be a QUEUE of the type declared in `ICEVENTS.INC` as follows:

```
EventActionQueue  QUEUE,TYPE
EventNo           SIGNED
Action            BYTE
END
```

Return Data Type:

**STRING**

## Example:

```
MyWebJavaListClass.CreateParams PROCEDURE(*HtmlClass Target)
Properties                        ANY
CurFont                         HtmlFontClass
Feq                              SIGNED,AUTO

CODE
Feq = SELF.Feq
IF (Feq{PROP:column}<>0) THEN Properties = Properties & ',COLUMN'.
IF (Feq{PROP:imm}) THEN      Properties = Properties & ',IMM'.
IF (Feq{PROP:vcr}) THEN      Properties = Properties & ',VCR'.
SELF.GetFont(CurFont)

SELF.CreateColorParameters(Target, SELF.AutoSpotLink)
Target.WriteAppletParameter('formatString', SELF.Feq{PROP:format})
Target.WriteAppletParameter('Events', |
    JavaEvents.GetEventString(SELF.EventActionQ, Update:OnBrowser))
Target.WriteAppletParameter('Properties', SUB(Properties, 2, -1))
IF ((SELF.Container &= NULL) OR (Target.GetFontChanged(CurFont)))
    Target.WriteAppletFontParameter(CurFont)
END
```

See Also: [WebJavaListClass.EventActionQ](#)

# WEB FILES CLASS

<b>Overview</b>	<b>489</b>
WebFilesClass Concepts .....	489
Relationship to Other Internet Builder Classes .....	489
Internet Connect Template Implementation .....	489
Source Files .....	490
<b>WebFilesClass Methods</b>	<b>491</b>
Functional Organization—Expected Use .....	491
GetAlias (return HTML alias for file) .....	492
GetDirectory (return temporary folder name) .....	493
GetFilename (return temporary file pathname) .....	494
GetProgramRef (return HTML program reference) .....	495
GetPublicDirectory (return public folder name) .....	496
GetRelativeFilename (return filename for Broker) .....	498
GetSeparateSecure (return web server security support) .....	499
GetTempFilename (return filename for Server) .....	500
Init (initialize the WebFilesClass object) .....	501
Kill (shut down the WebFilesClass object) .....	502
LoadImage (return linked image filename) .....	503
RemoveAll (remove temporary files and folders) .....	504
SelectTarget (set public or secure channel) .....	505



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### WebFilesClass Concepts

---

Typically your Web-enabled application instantiates a single WebFilesClass object to manage all the filenames, pathnames, directories, aliases, etc. referenced during a session.

There are many different types of files that may be referenced during a session, and the location (pathname) of the files depends on a number of factors, including the file type/content (.CAB, .ZIP, .CLASS, .HTM, .JSL, .ICO, .GIF, .JPG, and other image files, etc.), the Application Broker type (.EXE or .DLL), the secure/public status of the transmission, and any aliases set by the web-server software (such as Microsoft's Internet Information Server). See *Application Design Considerations—Directory Structures* in the *Internet Connect User's Guide* for more information.

The WebFilesClass object manages all the pathname information for the temporary and permanent files, directories, and aliases, including switching between public and secure channels as requested by the WebWindowClass.

### Relationship to Other Internet Builder Classes

---

The BrokerClass, WebServerClass, HtmlClass, JslManagerClass, and HttpClass all rely on the WebFilesClass to supply the correct pathnames for various files referenced during the session.

The WebFilesClass does not rely on any other classes.

### Internet Connect Template Implementation

---

The Internet Connect Templates generate code to instantiate a single global WebFilesClass object per application. The WebFilesClass object is called WebFilesManager and is referenced by many of the other IBC Library objects.

## Source Files

---

The WebFilesClass source code is installed by default to the \LIBSRC folder. The specific class declarations reside in the following files and their method definitions reside in the corresponding .CLW files.

ICFILES.INC      WebFilesClass

## *WebFilesClass Methods*

The WebFilesClass contains the methods listed below.

### **Functional Organization—Expected Use**

---

As an aid to understanding the WebFilesClass, it is useful to organize its various methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### **Primary Interface Methods**

The other IBC Library objects (WebBrokerClass, WebServerClass, HtmlClass, and JslManagerClass) call the primary interface methods fairly routinely. These primary interface methods can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize WebFilesClass object
Kill	shut down WebFilesClass object

##### **Mainstream Use:**

GetAlias	return HTML alias for file
----------	----------------------------

##### **Occasional Use:**

GetDirectory	return temporary folder name
GetPublicDirectory	return temporary folder name
GetFilename	return temporary file pathname
GetProgramRef	return HTML program reference
GetRelativeFilename	return filename for Broker
GetTempFilename	return filename for Server
LoadImage	return linked image filename
SelectTarget	set public or secure channel
RemoveAll	remove temporary files and folders
GetSeparateSecure	return web server security support

#### **Virtual Methods**

The WebFilesClass contains no virtual methods.

## GetAlias (return HTML alias for file)

### GetAlias( *filename* ), STRING

**GetAlias** Returns the HTML alias for the specified *filename*.  
*filename* A string constant, variable, EQUATE, or expression containing the (Windows) filename for which to return the HTML alias. This may be a full pathname or a filename with no path. If the path is omitted, GetAlias searches for the file in the directories available to the Server.

The **GetAlias** method returns the HTML alias for the specified *filename*. The HTML alias is the name by which the Client browser refers to the file. Various IBC methods call GetAlias to provide the correct filename when generating HTML to send to the Client browser.

The *filename* may be a full pathname or a filename with no path. If the path is omitted, GetAlias searches for the filename in the directories available to the Server.

GetAlias is required because the Server is running locally and refers to files by their Windows pathnames (e.g., c:\cwic serv\topspeed\customer.tps), while the Client is running remotely and must refer to the same files by their alias names (e.g., topspeed/customer.tps). Typically, aliases are established and maintained by the internet server software such as Microsoft Personal Web Server or Internet Information Server.

Return Data Type: **STRING**

Example:

```
HtmIClass.WriteAppletFilenameParameter PROCEDURE(STRING param, STRING Filename)
AliasName CSTRING(FILE:MaxFilename)
CODE
IF (Filename)
AliasName = SELF.Files.GetAlias(Filename)
SELF.WriteLine('<<param name=' & param & ' value="' & |
IC:QuoteText(AliasName,IC:RESET:Value) & '">')
END
```

## GetDirectory (return temporary folder name)

### GetDirectory( [*security*] ), STRING

**GetDirectory** Returns a directory name for temporary files.  
*security* An integer constant, variable, EQUATE, or expression indicating whether a public or a secure channel is required. If omitted, *security* defaults to the current security status, either Secure:None or Secure:Full.

The **GetDirectory** method returns a directory name with the specified *security*, (public or secure). Various IBC Library objects use the returned directory to hold temporary files used during this session.

The RemoveAll method deletes the temporary directory and its files.

Implementation:

EQUATEs for the *security* parameter are declared in ICFILES.INC as follows:

```
ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END
```

Return Data Type: **STRING**

Example:

```
WebFilesClass.RemoveAll PROCEDURE
CurIndex SIGNED,AUTO
CODE
IF (SELF.UniqueId)
  RemoveDirectory(SELF.GetDirectory(Secure:None))
IF (SELF.GetSeparateSecure())
  RemoveDirectory(SELF.GetDirectory(Secure:Full))
END
END
```

See Also: **RemoveAll**

## GetFilename (return temporary file pathname)

**GetFilename( *content* [,*security*] ), STRING**

**GetFilename** Returns a temporary file pathname.

*content* An integer constant, variable, EQUATE, or expression indicating the file's contents. The filename and extension depends on the *content*.

*security* An integer constant, variable, EQUATE, or expression indicating whether a public or a secure channel is required. If omitted, *security* defaults to the current security status, either Secure:None or Secure:Full.

The **GetFilename** method returns a temporary file pathname of the specified *security* (public or secure), for use during this session. Various IBC Library objects use this method to reference temporary files used during this session.

The *content* parameter determines the filename and extension. The *security* parameter determines the file's path.

Implementation: The GetFilename method calls the GetDirectory method to get the path.

EQUATES for the *content* parameter are declared in ICFILES.INC as follows:

```

ITEMIZE(1),PRE(Content)
Html      EQUATE
Js1       EQUATE
Unauthorized EQUATE
Last      EQUATE(Content:Unauthorized)
END

```

EQUATES for the *security* parameter are declared in ICFILES.INC:

```

ITEMIZE,PRE(Secure)
Default   EQUATE
None      EQUATE
Full      EQUATE
Last      EQUATE(Secure:Full)
END

```

Return Data Type: **STRING**

Example:

```

MyWebFilesClass.RemoveAll PROCEDURE
CODE
IF (SELF.UniqueId)
  RemoveDirectory(SELF.GetDirectory(Dir:Public))
  IF (SELF.GetSeparateSecure())
    RemoveDirectory(SELF.GetDirectory(Dir:Secure))
  . .

```

See Also: **GetDirectory**

## GetProgramRef (return HTML program reference)

**GetProgramRef**( [*security*] ), STRING

**GetProgramRef** Returns the HTML reference to the Server program.  
*security* An integer constant, variable, EQUATE, or expression indicating whether a public or a secure channel is required. If omitted, *security* defaults to the current security status, either Secure:None or Secure:Full.

The **GetProgramRef** method returns the HTML reference to the Server program. Various IBC objects use this method to generate HTML so the Client browser can send requests back to the Server.

Implementation: EQUATEs for the *security* parameter are declared in ICFILES.INC:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

Return Data Type: STRING

Example:

```

WebClientAreaClass.CreateHtml      PROCEDURE(*HtmlClass Target)
CurFont      HtmlFontClass
CODE
SELF.PushFont(Target)
SELF.OwnerWindow.CreateChildHtml(Target, FEQ:ClientArea)
SELF.PopFont(Target)
IF (SELF.OwnerWindow.IsSplash)
  Target.Write('<<CENTER>')
  Target.Write('<<A HREF="" & SELF.OwnerWindow.Files.GetProgramRef() &|
              "">' & SplashContinueText & '<</A>')
  Target.WriteLine('<</CENTER>')
POST(EVENT:CloseWindow)
END

```

## GetPublicDirectory (return public folder name)

---

**GetPublicDirectory( [*security*] ), STRING**

### GetPublicDirectory

Returns the public directory set by the Init method.

*security* An integer constant, variable, EQUATE, or expression indicating whether a public or a secure channel is required. If omitted, *security* defaults to the current security status, either Secure:None or Secure:Full.

The **GetPublicDirectory** method returns the public directory set by the Init method. If no public directory is specified by Init, the GetPublicDirectory method returns the public directory created by the Application Broker install program. See *Application Design Considerations—Directory Structures* in the *Internet Connect User's Guide* for more information.

Implementation:

EQUATEs for the *security* parameter are declared in ICFILES.INC as follows:

```
ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END
```

The Application Broker install program creates a subdirectory below the broker directory for delivering files to the Client. By default this directory is named \public; however, you may name this directory whatever you choose. This directory is used to deliver files (such as the Java Support Library, graphics or other HTML files) by the Application Broker.

Return Data Type:

STRING

Example:

```
WebWindowClass.TakeCreatePage      PROCEDURE
Client                             &WebClientManagerClass
Filename                           CSTRING(FILE:MaxFilePath),AUTO
CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetSecurity())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
    Filename = SELF.Files.GetFilename(Content:Html)
    IF (SELF.Server.GetRequestedWholePage())
        Client.NextHtmlPage
        SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
    ELSE
        Filename = SELF.Files.GetPublicDirectory() & 'dummy.htm'
        SELF.CreateDummyHtmlPage(SELF.HtmlTarget, Filename)
    END
    Client.TakeHtmlPage(Filename, SELF.GetTargetSecurity(), FALSE)
ELSE
    Client.Js1.OpenChannel(SELF.GetTargetSecurity(), SELF.Files)
    SELF.CreateJs1Data(Client.Js1)
    Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()
END
```

See Also:           **Init**

## GetRelativeFilename (return filename for Broker)

**GetRelativeFilename( *filename* ), STRING**

### GetRelativeFilename

Returns the filename by which the Application Broker knows the specified *filename*.

*filename* A string constant, variable, EQUATE, or expression containing the (Windows) filename for which to return the alternative filename. This may be a full pathname or a filename with no path. If the path is omitted, GetRelativeFilename searches for the file in the directories available to the Server.

The **GetRelativeFilename** method returns the filename by which the Application Broker references the specified *filename*. The Broker is unaware of its parent directories, so those components of the *filename* are removed. The BrokerClass calls GetRelativeFilename to provide the correct filename when communicating with the Application Broker.

The *filename* may be a full pathname or a filename with no path. If the path is omitted, GetRelativeFilename searches for the filename in the directories available to the Server.

GetRelativeFilename is required because the Server refers to files by their Windows pathnames (e.g., c:\cwcserv\topspeed\customer.tps), while the Broker is unaware of its parent directories and must refer to the same files by their relative names (e.g., topspeed/customer.tps).

Return Data Type: **STRING**

Example:

```
BrokerClass.TakeFile PROCEDURE(STRING Filename, SIGNED Security, BYTE dontmove)
Command             CSTRING(FILE:MaxFilePath)
IsAbsolute          BYTE
Flags               SIGNED(0)
CODE
Command = SELF.Files.GetRelativeFilename(Filename)
IsAbsolute = IC:IsAbsoluteURL(Command)
IF NOT (IsAbsolute OR DontMove)
    Command = IC:TranslateFilename(Command)
END
IF IsAbsolute
    Flags += RPC:AbsRedirect
ELSIF (dontmove)
    Flags += RPC:NoMove
END
IF (Security = Secure:Full)
    Flags += RPC:Secure
END
IC:SendPage(Command, Flags)
```

## GetSeparateSecure (return web server security support)

---

### GetSeparateSecure( *filename* ), **STRING**

The **GetSeparateSecure** method returns a value indicating whether the internet server software (such as Microsoft Personal Web Server or Internet Information Server) supports separate secure directories for secure channel transmissions. A return value of one (1) indicates separate secure channel directories; a value of zero (0) indicates a shared directory.

Return Data Type:      **BYTE**

Example:

```
WebFilesClass.RemoveAll                      PROCEDURE
CurIndex                      SIGNED, AUTO
CODE
IF (SELF.UniqueId)
    RemoveDirectory(SELF.GetDirectory(Secure:None))
    IF (SELF.GetSeparateSecure())
        RemoveDirectory(SELF.GetDirectory(Secure:Full))
END
END
```

## GetTempFilename (return filename for Server)

---

**GetTempFilename( *filename* ), STRING**

**GetTempFilename** Returns a temporary filename for use by the Server.

*filename*            A string constant, variable, EQUATE, or expression containing the (Windows) filename for which to return the alternative filename. This may be a full pathname or a filename with no path. If the path is omitted, GetTempFilename searches for the file in the directories available to the Server.

The **GetTempFilename** method returns a temporary filename for use by the Server during the current session.

The *filename* may be a full pathname or a filename with no path. If the path is omitted, GetTempFilename searches for the filename in the directories available to the Server.

Return Data Type:        **STRING**

Example:

```
HttpClass.Start PROCEDURE |
    (*HttpPageBaseClass HttpPage, SIGNED Status, <STRING Filename>)
    CODE

    SELF.ClearUp()
    SELF.HttpPage &= HttpPage
    SELF.HttpPage.Init(SELF, status, Filename, SELF.Files.GetTempFilename(Filename))
    SELF.HttpPage.PreparePage()
```

## Init (initialize the WebFilesClass object)

**Init**( *longfilenames* [, *subdirectory*] )

<b>Init</b>	Initializes the WebFilesClass object.
<i>longfilenames</i>	A Boolean constant, variable, EQUATE, or expression indicating whether the Server generates long filenames for its temporary files. Clarion versions 2.003 and lower ignore this parameter in favor of short filenames
<i>subdirectory</i>	A string constant, variable, EQUATE, or expression containing a partial pathname. The WebFilesClass uses this path fragment to construct pathnames for temporary files and folders for this session. If omitted, the temporary filenames and folder names are constructed without the path fragment.

The **Init** method initializes the WebFilesClass object.

The *subdirectory* parameter contains a pathname fragment that gives you some control over the placement of temporary files generated by the Server. The *subdirectory* parameter should not contain a drive specification such as c: or d:. The *subdirectory* parameter becomes the middle portion of generated pathnames. The WebFilesClass object automatically generates the beginning and ending of the pathnames, which comprise a sufficient name so that the *subdirectory* parameter may be omitted with no adverse effects.

See *Application Design Considerations—Directory Structures* in the *Internet Connect User's Guide* for more information on directories.

**Implementation:** The Init method sets the values of various filenames, paths, directories, and aliases used during this session.

**Example:**

```

Orders PROGRAM
!data declarations
WebFilesManager WebFilesClass !declare WebFilesManager object
CODE
WebFilesManager.Init(1,'Orders\Temp') !initialize WebFilesManager object
!program code
WebFilesManager.Kill

```

## Kill (shut down the WebFilesClass object)

---

### Kill

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Implementation: The **Kill** method is one of the few IBC methods you will typically call directly from your Web-enabled application.

Example:

```
Orders PROGRAM
!data declarations
WebFileManager WebFilesClass           !declare WebFileManager object
CODE
WebFileManager.Init(1,'Orders\Temp')    !initialize WebFileManager object
!program code
WebFileManager.Kill
```

## LoadImage (return linked image filename)

---

**LoadImage**( *filename* ), **STRING**  
| *control, property* |

<b>LoadImage</b>	Returns just the filename portion of a runtime property, such as PROP:Icon or PROP:Iconlist.
<i>filename</i>	A string variable or expression containing the return value of the runtime property.
<i>control</i>	An integer constant, variable, EQUATE or expression identifying a control (such as an IMAGE or LIST) with an associated image file or files. This is the control's field equate number.
<i>property</i>	An integer constant, variable, EQUATE or expression indicating the runtime property that identifies the image file.

The **LoadImage** method returns just the filename portion of a runtime property, such as PROP:Icon or PROP:Iconlist. The WebFilesClass uses this filename to construct appropriate pathnames for images the Server extracts to separate files for use by the Client browser. Typically, these are images that are linked in to the Server executable and are not available to the Client browser until they are written to a separate file.

Return Data Type: **STRING**

Example:

```
WebJavaButtonClass.GetFilename FUNCTION  
CODE  
RETURN SELF.OwnerWindow.Files.LoadImage(SELF.Feq, PROP:icon)
```

## RemoveAll (remove temporary files and folders)

---

### RemoveAll

The **RemoveAll** method removes all temporary directories and files created during the life of the `WebFilesClass` object.

Example:

```
WebServerClass.Kill PROCEDURE
```

```
CODE
SELF.Files.RemoveAll
IF (SELF.InRequest)
  IF (SELF.PageToReturnTo)
    SELF.Broker.TakeFile(SELF.PageToReturnTo, Dir:None, FALSE)
  ELSE
    SELF.Broker.TakeFile(SELF.Files.GetProgramRef(), Dir:None, FALSE)
  END
END
YIELD()
END
SELF.Broker.CloseChannel
SELF.Active = FALSE
```

## SelectTarget (set public or secure channel)

### SelectTarget( *security* )

<b>SelectTarget</b>	Puts the WebFilesClass object into public channel or secure channel mode.
<i>security</i>	An integer constant, variable, EQUATE, or expression indicating whether a public or a secure channel is required. Valid values are Secure:Full and Secure:None.

The **SelectTarget** method puts the WebFilesClass object into public channel mode or secure channel mode so that various WebFilesClass methods generate appropriate filenames and pathnames.

Implementation:

EQUATEs for the *security* parameter are declared in ICFILES.INC as follows:

```

ITEMIZE,PRE(Secure)
Default EQUATE
None EQUATE
Full EQUATE
Last EQUATE(Secure:Full)
END

```

Example:

```

MyWebWindowClass.TakeCreatePage PROCEDURE
Client &WebClientManagerClass
Filename CSTRING(FILE:MaxFilePath),AUTO
CODE
Client &= SELF.Server.Client
SELF.Files.SelectTarget(SELF.GetTargetDirType())
IF (SELF.Server.GetSendWholePage() OR NOT SELF.SentHtml)
Filename = SELF.Files.GetFilename(Content:Html)
IF (SELF.Server.GetRequestedWholePage())
Client.NextHtmlPage
SELF.CreateHtmlPage(SELF.HtmlTarget, Filename)
ELSE
SELF.CreateDummyHtmlPage(SELF.HtmlTarget, Filename)
END
Client.TakeHtmlPage(Filename, SELF.GetTargetDirType(), FALSE)
ELSE
Client.Js1.OpenChannel(SELF.GetTargetDirType(), SELF.Files)
SELF.CreateJs1Data(Client.Js1)
Client.Js1.CloseChannel
END
SELF.Server.TakePageSent()

```



# LAYOUT HTML CLASS

<b>Overview</b>	<b>509</b>
LayoutHtmlClass Concepts .....	509
Relationship to Other Internet Builder Classes .....	509
Internet Connect Template Implementation .....	510
Source Files .....	510
Conceptual Example .....	510
<b>LayoutHtmlClass Properties</b>	<b>511</b>
SnapX (horizontal grid snap) .....	511
SnapY (vertical grid snap) .....	511
Style (HTML table style).....	511
<b>LayoutHtmlClass Methods</b>	<b>512</b>
Functional Organization—Expected Use .....	512
CreateHtml (generate HTML table) .....	513
Init (initialize the LayoutHtmlClass object) .....	514
Insert (add control to layout process) .....	515
Kill (shut down the LayoutHtmlClass object) .....	516
SetCell (set current control/cell).....	517



## Overview

The `LayoutHtmlClass` generates an HTML `<TABLE>` structure to represent some portion of a Clarion WINDOW, typically a control and its child controls. Another object, such as a `WebWindowClass` object, can use the `LayoutHtmlClass` to generate an HTML Web page that closely mimics the appearance and behavior of a Clarion WINDOW.

## LayoutHtmlClass Concepts

---

The `LayoutHtmlClass` allows another object to generate *nested* HTML `<TABLE>` structures. By using the `LayoutHtmlClass` to nest HTML `<TABLE>` structures, an object such as a `WebWindowClass` object can produce a Web page that preserves the parent/child relationships (plus any inherited positions and properties that result from those relationships) that exist in the original WINDOW.

For example, a `WebWindowClass` object can use the `LayoutHtmlClass` to generate an HTML `<TABLE>` to represent an OPTION control and its RADIO controls, within another HTML `<TABLE>` containing the SHEET control that contains both the OPTION control and yet another control, such as a LIST.

In addition, by using the `LayoutHtmlClass` to nest HTML `<TABLE>` structures, another object, such as a `WebWindowClass` object, can align controls, horizontally and vertically, in a Windows-like manner, again, preserving the look and feel of the original Clarion WINDOW or REPORT.

## Relationship to Other Internet Builder Classes

---

The `WebWindowClass` and the `WebReportClass` create and manage `LayoutHtmlClass` instances as needed to optimally format all or part of a Web page.

The `LayoutHtmlClass` uses the `HtmlClass` to write the HTML code it specifies.

The `LayoutHtmlClass` uses the `HtmlItemClass/WebControlClass` to provide information about the WINDOW control it represents and to generate HTML code for the control within a specific HTML `<TABLE CELL>` designated by the `LayoutHtmlClass`.

## Internet Connect Template Implementation

---

The `WebWindowClass` and the `WebReportClass` create and manage `LayoutHtmlClass` instances as needed. Therefore the template generated code does not directly reference the `LayoutHtmlClass`.

## Source Files

---

The `LayoutHtmlClass` source code is installed by default to the `\LIBSRC` folder. The specific class declarations are in the following files and their method definitions are in the corresponding `.CLW` files.

ICLAYOUT.INC    `LayoutHtmlClass`

## Conceptual Example

---

The following example shows a typical sequence of statements to declare, instantiate, initialize, use, and terminate a `LayoutHtmlClass` object and related objects.

This example uses the `LayoutHtmlClass` to generate an HTML `<TABLE>` to represent a related list of `WINDOW` controls. Typically, the list of controls consists of all the children of a parent control. At the ultimate level, the `WINDOW` is the parent.

```
WebControlListClass.CreateHtml  PROCEDURE(*WebControlQueue Source, |
    *HtmlClass Target, STRING style, SIGNED SnapX, SIGNED SnapY)

Layout                          LayoutHtmlClass                !declare Layout object
CODE
Layout.Init(style, SnapX, SnapY)                                !initialize Layout object
SELF.AddControlsToLayout(Source, Layout)                        !identify controls to lay out
Layout.CreateHtml(Target)                                       !write HTML for the controls
Layout.Kill                                                       !shut down the Layout object

WebControlListClass.AddControlsToLayout  PROCEDURE|
    (*WebControlQueue Source, *LayoutHtmlClass Layout)

CurIndex                          SIGNED,AUTO
CODE
LOOP CurIndex = 1 TO RECORDS(Source)
    GET(Source, CurIndex)
    IF (Source.ThisControl.GetVisible())
        Layout.Insert(Source.ThisControl)
    END
END
END
```

## LayoutHtmlClass Properties

The LayoutHtmlClass contains the properties listed below.

### SnapX (horizontal grid snap)

---

#### SnapX SIGNED

The **SnapX** property contains a horizontal grid snap factor for aligning controls on the Web page. If the X coordinates of two WINDOW controls differ by less than the value of SnapX, the LayoutHtmlClass object left aligns the controls on the Web page.

Implementation: The Init method sets the value of the SnapX property. The Insert method aligns the controls by placing them in the same HTML <TABLE COLUMN>.

See Also: Init, Insert

### SnapY (vertical grid snap)

---

#### SnapY SIGNED

The **SnapY** property contains a vertical grid snap factor for aligning controls on the Web page. If the Y coordinates of two WINDOW controls differ by less than the value of SnapY, the LayoutHtmlClass object bottom aligns the controls on the Web page.

Implementation: The Init method sets the value of the SnapY property. The Insert method aligns the controls by placing them in the same HTML <TABLE ROW>.

See Also: Init, Insert

### Style (HTML table style)

---

#### Style CSTRING(100)

The **Style** property specifies the HTML <TABLE STYLE>.

Implementation: The Style property is primarily used to set the border width of the HTML <TABLE>. The Init method sets the value of the Style property.

See Also: Init

# ***LayoutHtmlClass Methods***

The LayoutHtmlClass contains the methods listed below.

## **Functional Organization—Expected Use**

---

As an aid to understanding the LayoutHtmlClass , it is useful to organize its methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

### **Primary Interface Methods**

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

#### **Housekeeping (one-time) Use:**

Init	initialize the LayoutHtmlClass object
CreateHtml	generate HTML table
Kill	shut down the LayoutHtmlClass object

#### **Mainstream (repeated) Use:**

Insert	add control to layout process
--------	-------------------------------

#### **Occasional Use:**

SetCell	set current control/cell
---------	--------------------------

### **Virtual Methods**

The LayoutHtmlClass has no virtual methods.

## CreateHtml (generate HTML table)

---

### CreateHtml( *html object* )

<b>CreateHtml</b>	Generates the HTML <TABLE> code.
<i>html object</i>	The label of the HtmlClass object that writes the specified HTML code.

The **CreateHtml** method generates the HTML <TABLE> code. Within the <TABLE> is HTML code representing each control (HtmlItemClass object) named by the Insert method.

Example:

```
WebControlListClass.CreateHtml PROCEDURE(*WebControlQueue Source, |  
    *HtmlClass Target, STRING style, SIGNED SnapX, SIGNED SnapY)
```

```
Layout                LayoutHtmlClass  
  
CODE  
Layout.Init(style, SnapX, SnapY)  
SELF.AddControlsToLayout(Source, Layout)  
Layout.CreateHtml(Target)  
Layout.Kill
```

See Also: **Insert**

## Init (initialize the LayoutHtmlClass object)

---

**Init**( *style*, *snapX*, *snapY* )

<b>Init</b>	Initializes the LayoutHtmlClass object.
<i>style</i>	A string constant, variable, EQUATE, or expression specifying the <STYLE> attributes for the HTML <TABLE>.
<i>snapX</i>	An integer constant, variable, EQUATE, or expression specifying the horizontal grid snap factor.
<i>snapY</i>	An integer constant, variable, EQUATE, or expression specifying the vertical grid snap factor.

The **Init** method initializes the LayoutHtmlClass object.

Example:

```
WebControlListClass.CreateHtml PROCEDURE(*WebControlQueue Source, |  
    *HtmlClass Target, STRING style, SIGNED SnapX, SIGNED SnapY)
```

```
Layout                LayoutHtmlClass  
  
CODE  
Layout.Init(style, SnapX, SnapY)  
SELF.AddControlsToLayout(Source, Layout)  
Layout.CreateHtml(Target)  
Layout.Kill
```

See Also:           **SnapX, SnapY, Style**

## Insert (add control to layout process)

---

### Insert( *control object* )

**Insert** Adds a control to the layout process.  
*control object* The label of the HtmlItemClass/WebControlClass object that represents a WINDOW or REPORT control.

The **Insert** method adds a control to the layout and HTML generation process. Each *control object* named by the Insert method is represented in the HTML <TABLE> structure generated by the CreateHtml method.

Example:

```
WebControlListClass.AddControlsToLayout PROCEDURE|
(*WebControlQueue Source, *LayoutHtmlClass Layout)

CurIndex          SIGNED,AUTO

CODE
LOOP CurIndex = 1 TO RECORDS(Source)
  GET(Source, CurIndex)
  IF (Source.ThisControl.GetVisible())
    Layout.Insert(Source.ThisControl)
  END
END
```

See Also: [CreateHtml](#)

## Kill (shut down the LayoutHtmlClass object)

---

### Kill

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Example:

```
WebControlListClass.CreateHtml PROCEDURE(*WebControlQueue Source, |  
    *HtmlClass Target, STRING style, SIGNED SnapX, SIGNED SnapY)
```

```
Layout                LayoutHtmlClass
```

```
CODE  
Layout.Init(style, SnapX, SnapY)  
SELF.AddControlsToLayout(Source, Layout)  
Layout.CreateHtml(Target)  
Layout.Kill
```

## SetCell (set current control/cell)

**SetCell( *x*, *y* ), LayoutCellClass, PROC**

<b>SetCell</b>	Sets the current control/cell for processing by other LayoutHtmlClass methods.
<i>x</i>	An integer constant, variable, EQUATE, or expression specifying the cell's column number.
<i>y</i>	An integer constant, variable, EQUATE, or expression specifying the cell's row number.

The **SetCell** method returns a reference to the specified control/cell (LayoutCellClass object) for processing by other LayoutHtmlClass methods. Cells are identified by their x(column) and y(row) coordinates.

This method has the PROC attribute so you can call it like a procedure and ignore the return value.

Return Data Type:      LayoutCellClass

Example:

```

LayoutHtmlClass.Kill                   PROCEDURE

CurCell           &LayoutCellClass,AUTO
CurItem           SIGNED,AUTO
Xpos               SIGNED,AUTO
Ypos               SIGNED,AUTO

CODE

IF (~SELF.Rows &= NULL)
  ASSERT(~SELF.RangeX.Bounds &= NULL)
  ASSERT(~SELF.RangeY.Bounds &= NULL)
  LOOP Ypos = 1 TO RECORDS(SELF.RangeY.Bounds)
    LOOP Xpos = 1 TO RECORDS(SELF.RangeX.Bounds)
      CurCell &= SELF.SetCell(Xpos, Ypos)
      CurCell.Kill
      DISPOSE(CurCell)
    END
  END
  DISPOSE(SELF.Rows.Columns)
END
DISPOSE(SELF.Rows)
SELF.RangeX.Kill
DISPOSE(SELF.RangeX)
SELF.RangeY.Kill
DISPOSE(SELF.RangeY)
END

```



# ***SUBMIT ITEM CLASS***

<b>Overview</b>	<b>521</b>
SubmitItemClass Concepts .....	521
Relationship to Other Internet Builder Classes .....	521
Internet Connect Template Implementation .....	521
Source Files .....	521
<b>SubmitItem Properties</b>	<b>522</b>
Event (event number) .....	522
Extra (other event information) .....	522
Freq (control number) .....	522
Name (new control contents) .....	523
NewValue (new control contents) .....	523
<b>SubmitItem Methods</b>	<b>524</b>
Reset (set SubmitItem properties) .....	524



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

## SubmitItemClass Concepts

---

### Browser Submit Strings

SubmitItemClass manages incoming data submitted from the Client browser. When the end user enters data through the browser, the browser sends a “submit string” to the Server. The submit string consists of a URL followed by a question mark (?) and a list of field assignments of the form ‘field=value&field2=value2&field3=value3.’ The SubmitItemClass represents a single field in this submit string. In addition to the new value entered in the browser, the SubmitItemClass object contains the field equate of the control and any event associated with the control (Event:Accepted, Event:ScrollUp, etc.).

The WebServerClass creates and manages SubmitItemClass instances as needed.

## Relationship to Other Internet Builder Classes

---

The WebServerClass creates and manages SubmitItemClass instances as needed to process the browser submit strings.

## Internet Connect Template Implementation

---

The WebServerClass creates and manages SubmitItemClass instances as needed. Therefore the template generated code does not directly reference SubmitItemClass objects.

## Source Files

---

The SubmitItemClass source code is installed by default to the \LIBSRC folder. The SubmitItemClass declarations reside in the following files and their method definitions reside in the corresponding .CLW files.

ICSERVER.INC    SubmitItemClass

# SubmitItem Properties

SubmitItemClass contains the properties listed below.

## Event (event number)

---

### Event SIGNED

The **Event** property contains the event number of the control updated in the client browser.

Implementation: The Reset method sets the value of the Event property. The WebControlClass uses this property to POST the appropriate event to the control in the Server (Web-enabled Clarion application).

See Also: Reset

## Extra (other event information)

---

### Extra ANY

The **Extra** property contains any other information associated with the Event. For example, the Extra property may contain the thumb position for a ScrollDrag event.

Implementation: The Reset method sets the value of the Extra property. The WebControlClass uses this property to update the control in the Server (Web-enabled Clarion application).

See Also: Reset

## Feq (control number)

---

### Feq SIGNED

The **Feq** property contains the control number of the control updated in the client browser.

Implementation: The Reset method sets the value of the Feq property. The WebControlClass uses this property to update the control in the Server (Web-enabled Clarion application).

See Also: Reset

## Name (new control contents)

---

Name	ANY
------	-----

The **Name** property contains the HTML field name.

Implementation: The Reset method sets the value of the Name property.

See Also: Reset

## NewValue (new control contents)

---

NewValue	ANY
----------	-----

The **NewValue** property contains the value entered into the client browser.

Implementation: The Reset method sets the value of the NewValue property. The WebControlClass uses this property to update controls in the Server (Web-enabled Clarion application).

For entry fields, the NewValue property typically contains a string value. For LISTS and SHEETS, NewValue typically contains a numeric positioning value.

See Also: Reset

## SubmitItem Methods

SubmitItemClass contains only one method.

### Reset (set SubmitItem properties)

---

#### Reset( *arguments* )

<b>Reset</b>	Sets the value of all the SubmitItemClass properties based on the value of <i>arguments</i> .
<i>arguments</i>	A string constant, variable, EQUATE, or expression containing the submit string from the client browser in the form <i>field=value</i>

The **Reset** method sets the value of all the SubmitItemClass properties by parsing the *arguments* parameter.

Implementation: The Reset method converts the browser (HTML) field name to a Clarion Field Equate number.

Example:

```

WebServerClass.SetNextAction  PROCEDURE

AssignPos          SIGNED
EndPos             SIGNED
EventNo           SIGNED
EventPos          SIGNED
StartPos          SIGNED
NIL               &SubmitItemClass

CODE

StartPos = SELF.ArgIndex

EndPos = INSTRING('&', SELF.Arguments, 1, StartPos)
IF (EndPos = 0)
    RETURN NIL
END

SELF.CurSubmit.Reset(SUB(SELF.Arguments, StartPos, EndPos-StartPos))

SELF.ArgIndex = EndPos + 1
RETURN SELF.CurSubmit

```

See Also: [WebServerClass.SetNextAction](#)

# HTML CLASS

<b>Overview</b>	<b>527</b>
HtmlClass Concepts .....	527
Relationship to Other Internet Builder Classes .....	527
Internet Connect Template Implementation .....	527
Source Files .....	528
<b>HtmlClass Properties</b>	<b>529</b>
AppletCount (Java applets on this HTML page) .....	529
Browser (browser manager object) .....	529
Client (client manager object) .....	529
Files (file manager object) .....	530
FirstControl (first input control) .....	530
FirstSelectable (first input control select all flag) .....	530
JavaLibraryCab (Java Support Library cabinet name) .....	531
JavaLibraryZip (Java Support Library zip name) .....	531
Option (web page scale information) .....	532
UseFonts (use Windows fonts on Web page) .....	532
<b>HtmlClass Methods</b>	<b>533</b>
Functional Organization—Expected Use .....	533
CreateOpen (start new HTML page) .....	535
GetFontChanged (return font changed flag) .....	536
GetFontStyle (return HTML STYLE string) .....	537
GetPixelsX (convert horizontal dialog units to pixels) .....	538
GetPixelsY (convert vertical dialog units to pixels) .....	539
GetControlReference (return control HREF) .....	540
Init (initialize the HtmlClass object) .....	541
Kill (shut down the HtmlClass object) .....	541
PopFont (restore pre-PushFont font) .....	542
PushFont (set current font) .....	543
TakeNewControl (set first control) .....	544
WriteAppletDimParameter (write Java applet dim parameter) .....	545
WriteAppletFilenameParameter (write Java applet filename parameter) .....	546
WriteAppletFontParameter (write Java applet font parameter) .....	547
WriteAppletFooter (end Java applet) .....	548
WriteAppletHeader (begin Java applet) .....	549

WriteAppletOptParameter (write Java applet parameter) .....	550
WriteAppletParameter (write Java applet parameter) .....	551
WriteAppletUAID (write Java applet unique Id) .....	552
WriteChildAppletFooter (end child Java applet) .....	553
WriteChildAppletHeader (begin child Java applet) .....	554
WriteContainerAppletHeader (begin container applet) .....	555
WriteControlFooter (end HTML control) .....	556
WriteControlHeader (begin HTML control) .....	556
WriteEventHandler (write HTML control accepted action) .....	557
WriteFontFooter (end HTML font) .....	559
WriteFontHeader (begin HTML font) .....	559
WriteFormFooter (end HTML FORM) .....	560
WriteFormHeader (begin HTML FORM) .....	560
WriteJavaScript (write shared JavaScript functions) .....	561
WriteOnFocusHandler (write HTML control selected action) .....	562
WriteRefreshTimer (write HTML timer refresh) .....	563
WriteSpace (write HTML space) .....	564
WriteSubmitApplet (write Java applet coordinator) .....	565
WriteTableFooter (end HTML TABLE) .....	566
WriteTableHeader (begin HTML TABLE) .....	566
WriteTableNewCol (begin HTML TABLE CELL) .....	567
WriteTableNewRow (begin HTML TABLE ROW) .....	567
WriteText (write breakable text string) .....	568

## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts* in the preceding chapter. This short topic contains information and terms that are prerequisite to the following material.

### HtmlClass Concepts

---

The HtmlClass class writes HTML and other source code specified by various IBC Library objects.

It contains functions for handling font information as HTML <STYLE> attributes and as Java applet parameters. It also contains functions for writing Java applets and JavaScript.

Typically your Web-enabled application instantiates a single global HtmlClass object.

### Relationship to Other Internet Builder Classes

---

The HtmlClass is derived from the TextOutputClass which provides basic text file handling, including creating, opening, writing, and closing a text file. See *Text Output Class* for more information.

The HtmlClass adds the HTML-specific text writing capability.

The WebWindowClass, the WebControlClass, and the WebReportClass all rely on the HtmlClass to write various snippets of HTML code to the appropriate destination.

### Internet Connect Template Implementation

---

The Internet Connect Templates generate code to instantiate a single global HtmlClass object. The object is called HtmlManager. Other IBC Library objects call the HtmlManager object as needed to write HTML snippets, so there are very few references to the HtmlManager object within the template generated code.

## Source Files

---

The HtmlClass source code is installed by default to the \LIBSRC folder. The specific class declarations reside in the following files and their method definitions reside in the corresponding .CLW files.

ICTXTOUT.INC    TextOutputClass

ICHTML.INC     HtmlClass

## HtmlClass Properties

The HtmlClass contains the properties listed below.

### AppletCount (Java applets on this HTML page)

---

**AppletCount**    **SIGNED**

The **AppletCount** property contains the number of Java applets on this HTML page. The HtmlClass does not use the AppletCount property; it is available for developer use.

### Browser (browser manager object)

---

**Browser**    **&BrowserManagerClass, PROTECTED**

The **Browser** property is a reference to the BrowserManagerClass object that provides information about the Client's Web browser, such as whether the browser supports Style Sheets or Java. The HtmlClass object uses this property to generate HTML appropriate for the Client's Web browser.

This property is PROTECTED, therefore, it can only be referenced by a HtmlClass method, or a method in a class derived from HtmlClass.

Implementation:    The CreateOpen method sets the value of the Browser property.

See Also:    CreateOpen

### Client (client manager object)

---

**Client**    **&WebClientManagerClass, PROTECTED**

The **Browser** property is a reference to the WebClientManagerClass object that provides information about the Client, such as the IP address. The HtmlClass object uses this property primarily to generate HTML control IDs.

This property is PROTECTED, therefore, it can only be referenced by a HtmlClass method, or a method in a class derived from HtmlClass.

Implementation:    The CreateOpen method sets the value of the Client property.

See Also:    CreateOpen

## Files (file manager object)

---

### Files    &WebFilesClass, PROTECTED

The **Files** property is a reference to the WebFilesClass object that provides all the file, directory, alias, and path information for the HtmlClass object. The HtmlClass object uses this property to supply appropriate filenames and pathnames within the generated HTML.

This property is PROTECTED, therefore, it can only be referenced by a HtmlClass method, or a method in a class derived from HtmlClass.

Implementation:        The Init method sets the value of the Files property.

See Also:                Init

## FirstControl (first input control)

---

### FirstControl    SIGNED

The **FirstControl** property contains the control number (field equate) of the control whose corresponding Web page control gets initial input focus when the browser displays the Web page.

The HtmlClass object uses this property to generate HTML code that mimics Windows behavior with regard to initial control focus on input forms.

Implementation:        The TakeNewControl method sets the value of the FirstControl property. The FirstControl property is only effective for entry fields and text fields under NetScape browsers.

See Also:                TakeNewControl

## FirstSelectable (first input control select all flag)

---

### FirstSelectable    BYTE

The **FirstSelectable** property indicates whether the control identified by the FirstControl property allows block selection of its text. A value of one (1) indicates block selectable text; a value of zero (0) indicates the text is not block selectable.

Implementation:        The TakeNewControl method sets the value of the FirstSelectable property. Entry fields are block selectable; text fields are not block selectable.

See Also:                FirstControl, TakeNewControl

## JavaLibraryCab (Java Support Library cabinet name)

---

JavaLibraryCab	CSTRING(FILE:MaxFilepath)
----------------	---------------------------

The **JavaLibraryCab** property contains the pathname of the Java Support Library cabinet (.CAB) file. The HtmlClass object uses this property to send the Java Support Library location to the Client's browser.

Implementation: The CreateOpen method sets the value of the JavaLibraryCab property.

See Also: Java Support Library

## JavaLibraryZip (Java Support Library zip name)

---

JavaLibraryZip	CSTRING(FILE:MaxFilepath)
----------------	---------------------------

The **JavaLibraryZip** property contains the pathname of the Java Support Library archive (.ZIP) file. The HtmlClass object uses this property to send the Java Support Library location to the Client's browser.

Implementation: The CreateOpen method sets the value of the JavaLibraryZip property.

See Also: Java Support Library

## Option (web page scale information)

---

### Option HtmlOptionGroup

The **Option** property contains horizontal and vertical scaling factors for the generated HTML page. The HtmlClass object uses this property to position Web page controls in proportion to their corresponding Window controls.

#### Implementation:

The CreateOpen method sets the value of the Option property. These scaling factors come from the Internet Options extension template Advanced tab: **HTML Scaling - Pixels Per Character**.

The GetPixelsX and GetPixelsY methods use this property to convert Window dialog units to Web page pixels.

The HtmlOptionGroup is declared in ICHTML.INC as follows:

```

HtmlOptionGroup    GROUP,TYPE
ScaleX             REAL
ScaleY             REAL
                  END

```

#### See Also:

CreateOpen, GetPixelsX, GetPixelsY

## UseFonts (use Windows fonts on Web page)

---

### UseFonts BYTE

The **UseFonts** property indicates whether to apply the fonts set in the Windows program to their corresponding HTML controls. A value of one (1) applies the Windows fonts; a value of zero (0) does not apply the Windows fonts, so that the Client's default fonts are used.

#### Implementation:

The Init method sets the UseFonts property to True. The UseFonts property is only effective if the Client's Web browser does not support style sheets.

#### See Also:

Init, BrowserManagerClass.SupportsStyleSheets

## HtmlClass Methods

The `HtmlClass` inherits all the methods from the `TextOutputClass` from which it is derived. See *TextOutputClass Methods* for more information.

In addition to (or instead of) the inherited methods, the `HtmlClass` contains the methods listed below.

### Functional Organization—Expected Use

---

As an aid to understanding the `HtmlClass`, it is useful to organize its various methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### Primary Interface Methods

The primary interface methods, which other IBC Library objects call fairly routinely, can be further divided into the following categories:

##### **Housekeeping (once per application) Use:**

Init	initialize the <code>HtmlClass</code> object
Kill	shut down the <code>HtmlClass</code> object

##### **Mainstream Use:**

##### **Occasional Use:**

CreateOpen	start new HTML page
GetFontChanged	return font changed flag
GetFontStyle	return HTML STYLE string
GetPixelsX	convert dialog units to pixels
GetPixelsY	convert dialog units to pixels
GetControlReference	return control HREF
PopFont	restore pre-PushFont font
PushFont	set current font
TakeNewControl	set first control
WriteAppletDimParameter	write Java applet dim parameter
WriteAppletFilenameParameter	write Java applet filename parameter
WriteAppletFontParameter	write Java applet font parameter
WriteAppletFooter	end Java applet
WriteAppletHeader	begin Java applet
WriteAppletOptParameter	write Java applet parameter
WriteAppletParameter	write Java applet parameter
WriteAppletUAID	write Java applet UAID parameter
WriteChildAppletFooter	end child Java applet
WriteChildAppletHeader	begin child Java applet
WriteContainerAppletHeader	begin container applet

WriteControlFooter	end HTML control
WriteControlHeader	begin HTML control
WriteEventHandler	write HTML control accepted action
WriteFontFooter	end HTML font
WriteFontHeader	begin HTML font
WriteFormFooter	end HTML FORM
WriteFormHeader	begin HTML FORM
WriteJavaScript	write shared JavaScript functions
WriteOnFocusHandler	write HTML control selected action
WriteRefreshTimer	write HTML timer refresh
WriteSpace	write HTML non-breaking space
WriteSubmitApplet	write shared Java submit applet
WriteTableFooter	end HTML TABLE
WriteTableHeader	begin HTML TABLE
WriteTableNewCol	begin HTML TABLE CELL
WriteTableNewRow	begin HTML TABLE ROW
WriteText	write breakable text string
Write <sup>1</sup>	write text string
WriteIn <sup>1</sup>	write end of line marker
Open <sup>1</sup>	open the file
GetSize <sup>1</sup>	return file size in bytes
Close <sup>1</sup>	close the file

<sup>1</sup> These methods are inherited from the `TextOutputClass`.

### **Virtual Methods**

Typically you will not call these methods directly—the Primary Interface methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

WriteJavaScript	write shared JavaScript functions
-----------------	-----------------------------------

## CreateOpen (start new HTML page)

**CreateOpen**( *pathname*, *scaling*, *JSL path*, *client* )

<b>CreateOpen</b>	Starts a new HTML page.
<i>pathname</i>	A string constant, variable, EQUATE, or expression containing the pathname of the file that contains the generated HTML code.
<i>scaling</i>	The label of the structure that contains horizontal and vertical Windows-to-Web scaling ratios.
<i>JSL path</i>	A string constant, variable, EQUATE, or expression containing the location of the Java Support Library.
<i>client</i>	The label of the WebClientManagerClass object that supplies information about the Client.

The **CreateOpen** method starts a new HTML page.

Implementation: The CreateOpen method sets the initial values of the Client, Browser, Option, AppletCount, JavaLibraryCab, and JavaLibraryZip properties. The CreateOpen method calls the TextOutputClass.CreateOpen method to initialize a new output file.

Example:

```
WebWindowClass.CreateHtmlPage PROCEDURE(*HtmlClass Target,STRING HtmlFilename)
CODE
Target.CreateOpen(HtmlFilename,SELF.HtmlOption,SELF.|           !start new HTML page
                  Server.JavaLibraryPath,SELF.Server.Client)
SELF.CreatePageHeader(Target)                               !page header
SELF.CreateChildHtml(Target,0,SELF.BorderWidth)           !page body
SELF.CreatePageFooter(Target)                              !page footer
Target.Close                                               !finish page
SELF.SentHtml = TRUE                                       !prepare for next page
```

See Also: **AppletCount**, **Browser**, **Client**, **Option**, **JavaLibraryCab**, **JavaLibraryZip**, **TextOutputClass.CreateOpen**

## GetFontChanged (return font changed flag)

**GetFontChanged**( *new font* ), BYTE

**GetFontChanged** Returns a value indicating whether the *new font* is different than the current Web page font.

*new font* The label of the HtmlFontClass object that contains the font information for a WINDOW or REPORT control.

The **GetFontChanged** method returns a value indicating whether the *new font* is different than the current Web page font. A return value of one (1) indicates the new font is different than the Web page font; a value of zero (0) indicates the fonts are the same, therefore no HTML is required to apply the new font.

The WebControlClass.GetFont method returns a reference that can be used for the *new font* parameter. The WebAreaClass.LocalFont method is a reference that can be used for the *new font* parameter.

Return Data Type:      **BYTE**

Example:

```
MyWebHtmlTabClass.CreateParams  PROCEDURE(*HtmlClass Target)
CurFont                        HtmlFontClass
ParentControl                   GROUP(WebControlRefGroup).

CODE
SELF.GetFont(CurFont)           !get current control font
IF (Target.GetFontChanged(CurFont))  !if different than Web page font
    Target.WriteAppletFontParameter(CurFont)  !apply new font to Web control
END
```

See Also:              WebControlClass.GetFont, WebAreaClass.LocalFont

## GetFontStyle (return HTML STYLE string)

**GetFontStyle**( *window font* [, *web font*] ), **STRING**

<b>GetFontStyle</b>	Returns the HTML code to implement the <i>window font</i> .
<i>window font</i>	The label of the HtmlFontClass object that contains the font information for a WINDOW or REPORT control.
<i>web font</i>	The label of the HtmlFontClass object that contains the current font information in effect for the Web page. If omitted, GetFontStyle supplies the current Web page font information.

The **GetFontStyle** method returns the HTML code needed to implement the *window font* on the generated Web page.

The WebControlClass.GetFont method returns a reference that can be used for either *font* parameter. The WebAreaClass.LocalFont method is a reference that can be used for either *font* parameter.

Return Data Type: **STRING**

Example:

```
MyWebAreaClass.GetCellAttributes  PROCEDURE(*HtmlClass Target)
Result  CSTRING(255)
CODE

IF (SELF.Background <> COLOR:None)          !add BGCOLOR attribute
    Result=Result&' BGCOLOR=''&IC:ColorText(SELF.Background)&''
END
IF (SELF.BackImage)                        !add BACKGROUND attribute
    Result=Result&' BACKGROUND=''&SELF.OwnerWindow.Files.GetAlias(SELF.BackImage)&''
END
RETURN Result&Target.GetFontStyle(SELF.LocalFont)    !add STYLE attribute
```

See Also: **WebControlClass.GetFont, WebAreaClass.LocalFont**

## GetPixelsX (convert horizontal dialog units to pixels)

### GetPixelsX( *width* ), SIGNED

**GetPixelsX** Returns the width in pixels for a given *width* in dialog units.

*width* An integer constant, variable, EQUATE, or expression containing a control width expressed in dialog units.

The **GetPixelsX** method returns the width in pixels for a given *width* in expressed in dialog units. The HtmlClass object uses this method to position Web page controls in proportion to their corresponding Window controls.

Implementation: The GetPixelsX method returns (*width* \* Option.ScaleX) + Option.ScaleX/2.

Return Data Type: SIGNED

Example:

```
HtmlClass.WriteBasicAppletHeader PROCEDURE|
    (STRING AppletName,STRING ClassName,SIGNED Width,SIGNED Height)

CODE
IF (Width = 0) THEN Width = 1.
IF (Height = 0) THEN Height = 1.
SELF.Write('<<applet NAME="'&AppletName&' CODEBASE="/' CODE='&ClassName&'.class')
SELF.Write(' ARCHIVE="' & SELF.JavaLibraryZip & '" MAYSCRIPT')
SELF.Write(' WIDTH=' & SELF.GetPixelsX(Width))           !convert width to pixels
SELF.Write(' HEIGHT=' & SELF.GetPixelsY(Height))        !convert height to pixels
SELF.WriteLine('>')
SELF.WriteAppletParameter('cabase',SELF.JavaLibraryCab)
SELF.WriteAppletParameter('USID',SELF.Files.GetProgramRef())
```

See Also: [Option](#)

## GetPixelsY (convert vertical dialog units to pixels)

### GetPixelsY( *height* ), SIGNED

**GetPixelsY** Returns the height in pixels for a given *height* in dialog units.

*height* An integer constant, variable, EQUATE, or expression containing a control height expressed in dialog units.

The **GetPixelsY** method returns the height in pixels for a given *height* in expressed in dialog units. The `HtmlClass` object uses this method to position Web page controls in proportion to their corresponding Window controls.

Implementation: The `GetPixelsY` method returns  $(height * Option.ScaleY) + Option.ScaleY/2$ .

Return Data Type: SIGNED

Example:

```
HtmlClass.WriteBasicAppletHeader PROCEDURE|
    (STRING AppletName,STRING ClassName,SIGNED Width,SIGNED Height)

CODE
IF (Width = 0) THEN Width = 1.
IF (Height = 0) THEN Height = 1.
SELF.Write('<<applet NAME="'&AppletName&' CODEBASE=/' CODE='&ClassName&'.class')
SELF.Write(' ARCHIVE="' & SELF.JavaLibraryZip & '" MAYSCRIPT')
SELF.Write(' WIDTH=' & SELF.GetPixelsX(Width))           !convert width to pixels
SELF.Write(' HEIGHT=' & SELF.GetPixelsY(Height))         !convert height to pixels
SELF.WriteLine('>')
SELF.WriteAppletParameter('cabbase',SELF.JavaLibraryCab)
SELF.WriteAppletParameter('USID',SELF.Files.GetProgramRef())
```

See Also: [Option](#)

## GetControlReference (return control HREF)

---

**GetControlReference( *control* ), STRING**

### GetControlReference

Returns the HTML HREF value for a control.

*control*

An integer constant, variable, EQUATE, or expression containing the control number. This is typically the field equate for the control.

The **GetControlReference** method returns the HTML HREF value for a control.

Implementation:

The HREF value for a control determines what the Client browser does when the end user manipulates or accepts the Web control. For example, the following HREF value causes the Client browser to send the string 'X30003=Ohio' to the Server at `http://ipaddress/path/WebApp.exe.0:`

```
HREF='http://ipaddress/path/WebApp.exe.0?X30003=Ohio'
```

The Server (your Web-enabled application) translates the X30003 to its corresponding Clarion control, assigns 'Ohio' to the control's USE variable, then POSTs an EVENT:Accepted for the control.

The **GetControlReference** method calls the **FilesClass.GetProgramRef** method to build the URL part of the HREF value.

Return Data Type:

STRING

Example:

```
WebHotlinkClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE
```

```
Target.WriteLine('<<NOBR>')
```

```
Target.Write('<<A HREF="' & Target.GetControlReference(SELF.Feq) & "'>')
```

```
Target.WriteLine(SELF.GetQuotedText() & '<</A>')
```

```
Target.WriteLine('<</NOBR><<BR>')
```

See Also:

**FilesClass.GetProgramRef**

## Init (initialize the HtmlClass object)

---

### Init( *files* )

<b>Init</b>	Initializes the HtmlClass object.
<i>files</i>	The label of the WebFilesClass object that supplies filenames, pathnames, directories, and aliases for the HtmlClass object.

The **Init** method initializes the HtmlClass object.

Implementation: The Init method sets the value of the Files property.

Example:

```
!data
CODE
WebFileManager.Init(1, '')
HtmlManager.Init(WebFileManager)
!program code
HtmlManager.Kill
WebFileManager.Kill
```

See Also: Files

## Kill (shut down the HtmlClass object)

---

### Kill

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Example:

```
!data
CODE
WebFileManager.Init(1, '')
HtmlManager.Init(WebFileManager)
!program code
HtmlManager.Kill
WebFileManager.Kill
```

## PopFont (restore pre-PushFont font)

---

### PopFont(*control*)

**PopFont**

Restores font information to its pre-PushFont state.

*control*

An integer constant, variable, EQUATE, or expression containing the control number of the control whose font is no longer the current one. This is typically the field equate for the control.

The **PopFont** method restores font information to its pre-PushFont state. Appropriate use of PushFont and PopFont allow child controls to correctly inherit or override parent control font information. Appropriate use simply means pairing a call to PushFont with a subsequent call to PopFont.

Example:

```
WebControlClass.PopFont PROCEDURE(*HtmlClass Target)
CODE
Target.PopFont(SELF.Feq)
```

See Also:

**PushFont**

## PushFont (set current font)

---

### PushFont( *font*, *control* )

<b>PushFont</b>	Makes the control's font the current one for use by other HtmlClass methods.
<i>font</i>	The label of the HtmlFontClass object that contains the font information for the <i>control</i> .
<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the control whose <i>font</i> to make current. This is typically the field equate for the control.

The **PushFont** method makes the control's font the current one for use by other HtmlClass methods. Appropriate use of PushFont and PopFont allow child controls to correctly inherit or override parent control font information. Appropriate use simply means pairing a call to PushFont with a subsequent call to PopFont.

Example:

```
WebControlClass.PushFont  PROCEDURE(*HtmlClass Target)
CurFont      HtmlFontClass
CODE
SELF.GetFont(CurFont)
Target.PushFont(CurFont, SELF.Feq)
```

See Also:           **PopFont**

## TakeNewControl (set first control)

**TakeNewControl**( *control*, *selectable* )

**TakeNewControl** Does any HtmlClass processing needed whenever a control is first encountered during HTML generation.

*control* An integer constant, variable, EQUATE, or expression containing the control number of the control being processed. This is typically the field equate for the control.

*selectable* An integer constant, variable, EQUATE, or expression indicating whether the control's text is block selectable.

The **TakeNewControl** method does any HtmlClass processing needed whenever a control is first encountered during HTML generation.

Implementation: The TakeNewControl method sets the FirstControl and FirstSelectable properties for Entry controls and Text controls only.

Example:

```
MyWebHtmlTextClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE
Target.TakeNewControl(SELF.Feq, FALSE)      !tell HtmlClass object a Text control
                                           !is being processed
Target.Write('<<TEXTAREA WRAP=VIRTUAL')
Target.Write(SELF.GetNameAttribute(Target))
Target.Write(' COLS="' & INT((SELF.Feq{PROP:width} + 2)/4) & "'')
Target.Write(' ROWS="' & INT((SELF.Feq{PROP:height} + 4)/8) & "'')
Target.Write('>')
Target.Write(IC:QuoteText(CLIP(CONTENTS(SELF.Feq)), IC:RESET:Text))
Target.WriteLine('<</TEXTAREA>')
SELF.UpdateCopyUse
```

See Also: FirstControl, FirstSelectable

## WriteAppletDimParameter (write Java applet dim parameter)

**WriteAppletDimParameter**( *x*, *y*, *width*, *height* )

### WriteAppletDimParameter

Writes the HTML to set an applet “Dim” parameter.

<i>x</i>	An integer constant, variable, EQUATE, or expression containing the applet’s horizontal position in dialog units.
<i>y</i>	An integer constant, variable, EQUATE, or expression containing the applet’s vertical position in dialog units.
<i>width</i>	An integer constant, variable, EQUATE, or expression containing the applet’s width in dialog units.
<i>height</i>	An integer constant, variable, EQUATE, or expression containing the applet’s height in dialog units.

The **WriteAppletDimParameter** method writes the HTML to set an applet “Dim” parameter. The “Dim” parameter determines the position and size coordinates for the applet.

Implementation: The **WriteAppletDimParameter** method calls the **WriteAppletParameter** method.

```
WebToolBarClass.CreateHtml      PROCEDURE(*HtmlClass Target)
!data
CODE
  Target.WriteContainerAppletHeader(SELF.Feq, SELF.GetAppletType(), MaxX, MaxY)
  IF (SameAppletType)
    Target.WriteAppletParameter('className', AppletType)
  END
  ControlNum = 1
  LOOP CurIndex = 1 TO RECORDS(Controls)
    GET(Controls, CurIndex)
    ThisControl &= Controls.ThisControl
    IF (ThisControl.GetVisible())
      ThisControl.GetPosition(x, y, width, height)
      Target.WriteChildAppletHeader('control' & ControlNum, ThisControl.Feq)
      Target.WriteAppletDimParameter(x, y, width, height)
      IF (NOT SameAppletType)
        Target.WriteAppletParameter('className', ThisControl.GetAppletType())
      END
      ThisControl.CreateParams(Target)
      Target.WriteChildAppletFooter
      ControlNum += 1
    END
  END
  Target.WriteAppletFooter
END
```

See Also: **WriteAppletParameter**

## WriteAppletFilenameParameter (write Java applet filename parameter)

**WriteAppletFilenameParameter**( *parameter*, *filename* )

### WriteAppletFilenameParameter

Writes the HTML to set an applet parameter equal to a pathname.

*parameter* A string constant, variable, EQUATE, or expression containing the name of the applet parameter to receive the *filename*.

*filename* A string constant, variable, EQUATE, or expression containing the filename to assign to the applet *parameter*.

The **WriteAppletFilenameParameter** method writes the HTML to set an applet parameter equal to a pathname.

Implementation: The **WriteAppletFilenameParameter** method expands the filename to its full pathname, then calls the **WriteAppletParameter** method.

Example:

```
WebJavaButtonClass.CreateParams  PROCEDURE(*HtmlClass Target)
Filename  CSTRING(FILE:MaxFileName)
CODE
Filename = SELF.GetFilename()

Target.WriteAppletFilenameParameter('Picture', Filename)
Target.WriteAppletOptParameter('Label', SELF.GetText())
Target.WriteAppletOptParameter('Hint', SELF.Feq{PROP:tooltip})
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
IF (SELF.GetEventAction(EVENT:Accepted) = Update:Full)
    Target.WriteAppletParameter('Submit', 1)
END
```

See Also: **WriteAppletParameter**

## WriteAppletFontParameter (write Java applet font parameter)

---

### WriteAppletFontParameter( *font* )

#### WriteAppletFontParameter

Writes the HTML to set an applet font parameter.

*font*                      The label of the HtmlFontClass object that contains the font information to apply.

The **WriteAppletFontParameter** method writes the HTML to set an applet font parameter.

Implementation:        The WriteAppletFontParameter method calls the WriteAppletParameter method.

Example:

```
WebJavaStringClass.CreateParams PROCEDURE(*HtmlClass Target)
CurFont                      HtmlFontClass
CODE
SELF.GetFont(CurFont)
SELF.LastText = SELF.GetText()
SELF.CreateColorParameters(Target, SELF.AutoSpotLink)
Target.WriteAppletOptParameter('Text', SELF.LastText)
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
Target.WriteAppletFontParameter(CurFont)
IF (SELF.CanBreak)
    Target.WriteAppletParameter('Wrap', '1')
END
IF (SELF.AutoSpotLink)
    Target.WriteAppletOptParameter('AutoSpotLink', SELF.AutoSpotLink)
END
```

See Also:                WriteAppletParameter

## WriteAppletFooter (end Java applet)

---

### WriteAppletFooter

The **WriteAppletFooter** method writes the HTML code (JavaScript) to end a Java applet started by the **WriteAppletHeader** method.

Example:

```
WebJavaStringClass.CreateHtml PROCEDURE(*HtmlClass Target)

Height          SIGNED,AUTO
Width           SIGNED,AUTO
CODE
GetPosition(SELF.Feq,,,Width,Height)
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

See Also: [WriteAppletHeader](#)

## WriteAppletHeader (begin Java applet)

**WriteAppletHeader**( *control* ,*class*, *width*, *height* )

### WriteAppletHeader

Writes the HTML code (JavaScript) to begin a Java applet.

<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Window control corresponding to the Java applet. This is typically the field <code>equate</code> for the control.
<i>class</i>	A string constant, variable, EQUATE, or expression containing the name of the Java Support Library (JSL) class instantiated by the applet.
<i>width</i>	An integer constant, variable, EQUATE, or expression containing the width of the Window control corresponding to the Java applet.
<i>height</i>	An integer constant, variable, EQUATE, or expression containing the height of the Window control corresponding to the Java applet.

The **WriteAppletHeader** method writes the HTML code (JavaScript) to begin a Java applet. End the Java applet with the **WriteAppletFooter** method.

Example:

```
WebJavaStringClass.CreateHtml PROCEDURE(*HtmlClass Target)
Height          SIGNED,AUTO
Width           SIGNED,AUTO
CODE
GetPosition(SELF.Feq,,Width,Height)
Target.WriteAppletHeader(SELF.Feq,SELF.GetAppletType(),Width,Height)
SELF.CreateParams(Target)
Target.WriteAppletFooter
```

See Also: **WriteAppletFooter**

## WriteAppletOptParameter (write Java applet parameter)

**WriteAppletOptParameter**( *parameter*, *value* )

### WriteAppletOptParameter

Writes the HTML to set an applet parameter.

*parameter*      A string constant, variable, EQUATE, or expression containing the name of the applet parameter to receive the *value*.

*value*            A string constant, variable, EQUATE, or expression containing the value to assign to the applet *parameter*.

The **WriteAppletOptParameter** method writes the HTML (JavaScript) to set an applet parameter.

Implementation:      The WriteAppletOptParameter method only writes the HTML if *value* is not null. The WriteAppletOptParameter method calls the WriteAppletParameter method.

Example:

```

WebJavaStringClass.CreateParams PROCEDURE(*HtmlClass Target)

CurFont                    HtmlFontClass
CODE

SELF.GetFont(CurFont)
SELF.LastText = SELF.GetText()

SELF.CreateColorParameters(Target, SELF.AutoSpotLink)
Target.WriteAppletOptParameter('Text', SELF.LastText)
Target.WriteAppletOptParameter('Align', SELF.GetAlignText())
Target.WriteAppletFontParameter(CurFont)
IF (SELF.CanBreak)
    Target.WriteAppletParameter('Wrap', '1')
END
IF (SELF.AutoSpotLink)
    Target.WriteAppletOptParameter('AutoSpotLink', SELF.AutoSpotLink)
END

```

See Also:              [WriteAppletParameter](#)

## WriteAppletParameter (write Java applet parameter)

**WriteAppletParameter**( *parameter*, *value* )

### WriteAppletParameter

Writes the HTML to set an applet parameter.

*parameter* A string constant, variable, EQUATE, or expression containing the name of the applet parameter to receive the *value*.

*value* A string constant, variable, EQUATE, or expression containing the value to assign to the applet *parameter*.

The **WriteAppletParameter** method writes the HTML (JavaScript) to set an applet parameter.

Implementation: Use the WriteAppletParameter method when *value* is not null. Use the WriteAppletOptParameter method when *value* may be null.

Example:

```
MyWebControlClass.CreateColorParameters PROCEDURE(*HtmlClass Target)
ForeColor          LONG,AUTO
BackColor          LONG,AUTO
CODE
GETFONT(SELF.Feq,,, ForeColor)
BackColor = SELF.GetBackgroundColor()
IF (ForeColor <> 0) AND (ForeColor <> COLOR:None)
    Target.WriteAppletParameter('ForeColor', IC:RGB(ForeColor))
END
IF (BackColor <> COLOR:None)
    Target.WriteAppletParameter('BackColor', IC:RGB(BackColor))
END
```

See Also: [WriteAppletOptParameter](#)

## WriteAppletUAID (write Java applet unique Id)

---

### WriteAppletUAID( *control* )

**WriteAppletUAID** Writes the HTML code to set an applet UAID parameter.

*control* An integer constant, variable, EQUATE, or expression containing the control number of the Window control corresponding to the Java applet. This is typically the field equate for the control.

The **WriteAppletUAID** method writes the HTML to set an applet UAID parameter. The UAID parameter uniquely identifies the applet. All communication with the applet uses this UAID value.

Implementation: The WriteAppletUAID method calls the WriteAppletParameter method.

Example:

```
HtmIClass.WriteAppletHeader PROCEDURE |  
  (SIGNED Freq,STRING AppletName,STRING ClassName,SIGNED Width,SIGNED Height)  
  CODE  
  SELF.WriteBasicAppletHeader(AppletName,'ClarionLoader',Width,Height)  
  SELF.WriteAppletParameter('className',ClassName)  
  SELF.WriteAppletUAID(Freq)
```

See Also: [WriteAppletParameter](#)

## WriteChildAppletFooter (end child Java applet)

---

### WriteChildAppletFooter

The **WriteChildAppletFooter** method writes the HTML code (JavaScript) to end a nested Java applet started by the **WriteChildAppletHeader** method within a container applet. A container applet can contain other Java applets. Container applets and their children require less HTML code and load quicker than a corresponding set of individual applets.

Example:

```
MyWebHtmlSheetClass.CreateTabControl PROCEDURE(*WebControlQueue TabControls,|
                                     *HtmlClass Target,SIGNED Alignment,SIGNED SelectedTabFeq)
!data declarations
CODE
NumRecords = RECORDS(TabControls)
IF (NumRecords > 0)
  SELF.PushFont(Target)
  TabNum = 1
  Target.WriteContainerAppletHeader(SELF.Feq, 'ClarionTabControl', MaxX, MaxY)
  LOOP CurIndex = 1 TO NumRecords
    GET(TabControls, CurIndex)
    ThisControl &= TabControls.ThisControl
    IF (ThisControl.GetVisible())
      Target.WriteChildAppletHeader('tab' & TabNum, ThisControl.Feq)
      ThisControl.CreateParams(Target)
      Target.WriteChildAppletFooter
      TabNum = TabNum + 1
    END
  END
  Target.WriteAppletFooter
  SELF.PopFont(Target)
END
```

See Also:

**WriteChildAppletHeader, WriteContainerAppletHeader**

## WriteChildAppletHeader (begin child Java applet)

**WriteChildAppletHeader**( *name*, *control* )

### WriteChildAppletHeader

Writes the HTML code to begin a nested Java applet.

*name* A string constant, variable, EQUATE, or expression containing the name of the applet.

*control* An integer constant, variable, EQUATE, or expression containing the control number of the Window control corresponding to the Java applet. This is typically the field equate for the control.

The **WriteChildAppletHeader** method writes the HTML code to begin a nested Java applet within a container applet. A container applet can contain other Java applets. Container applets and their children require less HTML code and load quicker than a corresponding set of individual applets.

End the nested Java applet with the **WriteChildAppletFooter** method.

Example:

```
MyWebHtmlSheetClass.CreateTabControl PROCEDURE(*WebControlQueue TabControls,|
                                     *HtmlClass Target,SIGNED Alignment,SIGNED SelectedTabFeq)
!data declarations
CODE
NumRecords = RECORDS(TabControls)
IF (NumRecords > 0)
  SELF.PushFont(Target)
  TabNum = 1
  Target.WriteContainerAppletHeader(SELF.Feq, 'ClarionTabControl', MaxX, MaxY)
  LOOP CurIndex = 1 TO NumRecords
    GET(TabControls, CurIndex)
    ThisControl &= TabControls.ThisControl
    IF (ThisControl.GetVisible())
      Target.WriteChildAppletHeader('tab' & TabNum, ThisControl.Feq)
      ThisControl.CreateParams(Target)
      Target.WriteChildAppletFooter
      TabNum = TabNum + 1
    END
  END
  Target.WriteAppletFooter
  SELF.PopFont(Target)
END
```

See Also: **WriteChildAppletFooter**, **WriteContainerAppletHeader**

## WriteContainerAppletHeader (begin container applet)

**WriteContainerAppletHeader**( *control*, *class*, *width*, *height* )

### WriteContainerAppletHeader

Writes the HTML code to begin a container applet.

<i>control</i>	An integer constant, variable, EQUATE, or expression containing the control number of the Window control corresponding to the Java applet. This is typically the field <code>equate</code> for the control.
<i>class</i>	A string constant, variable, EQUATE, or expression identify the JSL container class.
<i>width</i>	An integer constant, variable, EQUATE, or expression containing the width of the applet in dialog units.
<i>height</i>	An integer constant, variable, EQUATE, or expression containing the height of the applet in dialog units.

The **WriteContainerAppletHeader** method writes the HTML code to begin a container applet. A container applet can contain other Java applets. Container applets and their children require less HTML code and load quicker than a corresponding set of individual applets.

End the container applet with the **WriteAppletFooter** method.

Example:

```
MyWebHtmlSheetClass.CreateTabControl PROCEDURE(*WebControlQueue TabControls,|
    *HtmlClass Target,SIGNED Alignment,SIGNED SelectedTabFeg)
!data declarations
CODE
NumRecords = RECORDS(TabControls)
IF (NumRecords > 0)
    SELF.PushFont(Target)
    TabNum = 1
    Target.WriteContainerAppletHeader(SELF.Feg, 'ClarionTabControl', MaxX, MaxY)
    LOOP CurIndex = 1 TO NumRecords
        GET(TabControls, CurIndex)
        ThisControl &= TabControls.ThisControl
        IF (ThisControl.GetVisible())
            Target.WriteChildAppletHeader('tab' & TabNum, ThisControl.Feg)
            ThisControl.CreateParams(Target)
            Target.WriteChildAppletFooter
            TabNum = TabNum + 1
        END
    END
    Target.WriteAppletFooter
    SELF.PopFont(Target)
END
```

See Also: [WriteAppletFooter](#)

## WriteControlFooter (end HTML control)

---

### WriteControlFooter

The **WriteControlFooter** method writes the HTML code to end an HTML control started by the WriteControlHeader method.

Implementation: The WriteControlFooter method calls the WriteFontFooter method.

Example:

```
WebCaptionClass.CreateHtml PROCEDURE(*HtmlClass Target)

CODE
SELF.PushFont(Target)
Target.WriteControlHeader           !write HTML for caption control header
Target.Write(SELF.GetQuotedText())  !write HTML for caption control text
Target.WriteControlFooter           !write HTML for caption control footer
SELF.PopFont(Target)
```

See Also: WriteControlHeader, WriteFontFooter

## WriteControlHeader (begin HTML control)

---

### WriteControlHeader

The **WriteControlHeader** method writes the HTML code to begin an HTML control. End the control with the WriteControlFooter method.

Implementation: The WriteControlHeader method calls the WriteFontHeader method.

Example:

```
WebCaptionClass.CreateHtml PROCEDURE(*HtmlClass Target)

CODE
SELF.PushFont(Target)
Target.WriteControlHeader           !write HTML for caption control header
Target.Write(SELF.GetQuotedText())  !write HTML for caption control text
Target.WriteControlFooter           !write HTML for caption control footer
SELF.PopFont(Target)
```

See Also: WriteControlFooter, WriteFontHeader

## WriteEventHandler (write HTML control accepted action)

---

**WriteEventHandler**( *action*, *event handler*, *return value* )

### **WriteEventHandler**

Writes the HTML code that defines what happens when the end user changes the contents of a control with the Web browser.

*action* An integer constant, variable, EQUATE, or expression indicating the action to take when the change event occurs. Typically, the GetEventAction method supplies this value.

*event handler* A string constant, variable, EQUATE, or expression naming the browser JavaScript event handler associated with the HTML control.

*return value* A string constant, variable, EQUATE, or expression indicating the JavaScript parameter value to return to the Server for the updated control.

The **WriteEventHandler** method writes the HTML code that defines the handling for a Web page control when the end user changes the contents of the control with the Web browser.

Typically, the HTML code calls a JavaScript function that submits a request to the Server. The request includes the control's new value, an associated event code, and optionally, a request to refresh the HTML page.

#### Implementation:

The WriteEventHandler method defines a call to one of two JavaScript functions. If *action* is Update:Full, WriteEventHandler defines a call to the icSubmitForm() function; otherwise, WriteEventHandler defines a call to the changed() function, passing *action* as the first parameter.

The *action* parameter EQUATEs are in ICSTD.EQU as follows:

Update:OnBrowser	EQUATE(0)
Update:Partial	EQUATE(1)
Update:Full	EQUATE(2)
Update:Refresh	EQUATE(3)

Typically, the GetEventAction method supplies this value for the WriteEventHandler method. See the example.

The *event handler* parameter EQUATEs are in ICHTML.INC as follows:

```
HTML:EntryChanged    EQUATE('onChange')
HTML:TextChanged     EQUATE('onChange')
HTML:CheckChanged    EQUATE('onClick')
HTML:RadioChanged    EQUATE('onClick')
HTML:SelectChanged   EQUATE('onChange')
```

The *changed() parameter* parameter EQUATEs are in ICHTML.INC as follows:

```
HTML:EntryValue     EQUATE('this.value')           !control's value
HTML:TextValue      EQUATE('this.value')           !control's value
HTML:CheckValue     EQUATE('this.checked')         !control's on/off state
HTML:SelectValue    EQUATE('this.selectedIndex') !selected row/tab/radio
```

The “this” qualifier identifies the current control object for Java enabled browsers.

The WriteEventHandler method wrote just the black code in the following HTML code examples:

```
<TD><INPUT TYPE=TEXT VALUE="00003" NAME="X30003" SIZE="10"
onChange="icSubmitForm()" OnFocus="this.select()"></TD>

<TD><INPUT TYPE=TEXT VALUE="Dogwood Realty" NAME="X30005" SIZE="30"
onChange="changed(0,this.name,this.value)" OnFocus="this.select()"></TD>
```

This Server generated HTML code defines two different controls with different event handling behavior. On a change to the X30003 control, the browser executes the icSubmitForm() function; on a change to the X30005 control, the browser executes the changed() function.

Example:

```
MyWebHtmlCheckClass.CreateCellContents PROCEDURE(*HtmlClass Target)
CODE
Target.Write('<<INPUT TYPE=CHECKBOX VALUE="1"')
IF (SELF.Feq{PROP:checked})
    Target.Write(' CHECKED')
END
Target.Write(SELF.GetNameAttribute(Target))
Target.WriteEventHandler|
    (SELF.GetEventAction(EVENT:Accepted),HTML:CheckChanged,HTML:CheckValue)
Target.Write('>')
Target.Write(SELF.GetQuotedText())
```

See Also: [GetEventAction](#), [WriteJavaScript](#)

## WriteFontFooter (end HTML font)

---

### WriteFontFooter( *font* )

**WriteFontFooter** Writes the HTML code to end a font setting started by the WriteFontHeader method.

*font* The label of the HtmlFontClass object that contains the font information to apply.

The **WriteFontFooter** method writes the HTML code to end a font setting started by the WriteFontHeader method.

Example:

```
ReportStringClass.CreateHtml PROCEDURE(*HtmlClass H)
CODE
H.WriteFontHeader(SELF.Font)
H.WriteLine(SELF.Text)
H.WriteFontFooter(SELF.Font)
```

See Also: WriteFontHeader

## WriteFontHeader (begin HTML font)

---

### WriteFontHeader( *font* )

**WriteFontHeader** Writes the HTML code to begin a font setting.

*font* The label of the HtmlFontClass object that contains the font information to apply.

The **WriteFontHeader** method writes the HTML code to begin a font setting. End the font setting with the WriteFontFooter method.

Example:

```
HtmlClass.WriteControlHeader PROCEDURE
CODE
IF (NOT SELF.Browser.SupportsStyleSheets AND SELF.UseFonts)
  IF (RECORDS(SELF.Fonts) > 0)
    SELF.WriteFontHeader(SELF.Fonts.ThisFont)
  END
END
```

See Also: WriteFontFooter

## WriteFormFooter (end HTML FORM)

---

### WriteFormFooter

The **WriteFormFooter** method writes the HTML code to end an HTML FORM started by the WriteFormHeader method.

Example:

```
MyWebWindowClass.CreatePageFooter PROCEDURE(*HtmlClass Target)
CODE
Target.WriteSubmitApplet(SELF.TimerDelay*1000,SELF.TimerAction)
Target.WriteFormFooter                                !end the form
Target.WriteJavaScript
SELF.BodyFooter(Target)
Target.WriteLine('<</BODY>')
Target.WriteLine('<</HTML>')
```

See Also: [WriteFormHeader](#)

## WriteFormHeader (begin HTML FORM)

---

### WriteFormHeader( [*attr*] )

#### WriteFormHeader

Writes the HTML code to begin an HTML FORM.

*attr* A string constant, variable, EQUATE, or expression containing the attribute to the HTML FORM. If omitted, the method writes no attribute.

The **WriteFormHeader** method writes the HTML code to begin an HTML FORM. End the FORM with the WriteFormFooter method.

to begin the HTML FORM.

Example:

```
MyWebWindowClass.CreatePageHeader PROCEDURE(*HtmlClass Target)
CODE
Target.WriteLine('<<HTML>')
Target.WriteLine('<<HEAD>')
Target.Write('<<TITLE>')
SELF.TitleContents(Target)
Target.WriteLine('<</TITLE>')
Target.WriteLine('<</HEAD>')
Target.Write('<<BODY>')
Target.Write(' onLoad="setuptimer()" onUnload="killtimer()")')
Target.WriteLine('>')
Target.WriteFormHeader()                             !begin the form
SELF.BodyHeader(Target)
```

See Also: [WriteFormFooter](#)

## WriteJavaScript (write shared JavaScript functions)

### WriteJavaScript, VIRTUAL

The **WriteJavaScript** method writes several HTML (JavaScript) functions.

For Java enabled browsers, the HTML code that defines the Web page can execute these functions upon certain HTML events, such as `onLoad`, `onUnload`, `onFocus`, `onChange`, `onSubmit`, etc.

`WriteJavaScript` is a `VIRTUAL` method so that other base class methods can directly call the `WriteJavaScript` virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation:

The `WriteJavaScript` method generates the following JavaScript functions:

```
<SCRIPT LANGUAGE="JavaScript">
  function checkSubmit(form)
  function setuptimer()
  function setupapplet()
  function dotimer()
  function killtimer()
  function changed(i, name, value)
  function freeChangeList()
  function icSubmitForm()
  function reject(name)
  function hotlinkto(wher)
  function ShowHelp(page,style)
</SCRIPT>
```

HTML code can associate these functions with specific HTML events, so that the function is called when the event occurs. For example, the `WriteEventHandler` method associates either the `icSubmitForm()` or the `changed()` function with an “accepted” event for an HTML control.

Example:

```
MyWebWindowClass.CreatePageFooter PROCEDURE(*HtmlClass Target)
CODE
Target.WriteSubmitApplet(SELF.TimerDelay*1000, SELF.TimerAction)
Target.WriteFormFooter
Target.WriteJavaScript                               !write shared JavaScript functions
SELF.BodyFooter(Target)
Target.WriteLine('<</BODY>')
Target.WriteLine('<</HTML>')
```

See Also: [WriteEventHandler](#)

## WriteOnFocusHandler (write HTML control selected action)

### WriteOnFocusHandler( [*action*] )

#### WriteOnFocusHandler

Writes the HTML code that defines what happens when the end user selects the control.

*action* A string constant, variable, EQUATE, or expression containing the text of the function call to execute when the control is selected. If omitted, the browser calls the `this.select()` function.

The **WriteOnFocusHandler** method writes the HTML code that defines the handling for a Web page control when the end user selects the control with the Web browser.

Typically, the HTML code calls the `this.select()` function. The “this” qualifier identifies the current control object for Java enabled browsers. The `select()` function implements standard control selection behavior.

#### Implementation:

If *action* is null or omitted, **WriteOnFocusHandler** generates a call to the `this.select()` function; otherwise, **WriteOnFocusHandler** generates the function call defined within the *action* parameter.

The **WriteOnFocusHandler** method wrote just the black code in the following HTML code example:

```
<TD><INPUT TYPE=TEXT VALUE="00003" NAME="X30003" SIZE="10"
onChange="icSubmitForm()" OnFocus="this.select()"></TD>
```

This Server generated HTML code defines a control such that, when the control is selected (gains focus), the browser executes the `this.select()` function.

#### Example:

```
MyWebHtmlEntryClass.CreateCellContents PROCEDURE(*HtmlClass Target)
CODE
Target.TakeNewControl(SELF.Feq, TRUE)
Target.Write('<<INPUT TYPE=TEXT')
Target.Write(' VALUE="')
Target.Write(SELF.GetQuotedText())
Target.Write('')
Target.Write(SELF.GetNameAttribute(Target))
Target.Write(' SIZE="' & INT((SELF.Feq{PROP:width} + 2)/4) & "')
Target.WriteEventHandler|
    (SELF.GetEventAction(EVENT:Accepted),HTML:EntryChanged,HTML:EntryValue)
Target.WriteOnFocusHandler |set up gain focus handling
Target.WriteLine('>')
```

## WriteRefreshTimer (write HTML timer refresh)

---

### WriteRefreshTimer( *delay* )

#### WriteRefreshTimer

Writes the HTML code that defines an automatic timed page refresh.

*delay* An integer constant, variable, EQUATE, or expression containing the number of seconds to delay before refreshing the HTML page.

The **WriteRefreshTimer** method writes the HTML code that defines an automatic timed page refresh. The HTML code requests a new page from the same Server instance that generated the current page.

Implementation:

If *delay* is not zero (0), WriteRefreshTimer generates a META HTTP segment that defines the delay period and the action to take when the period expires. The WriteRefreshTimer method wrote just the black code in the following HTML code fragment:

```
<HTML>
<META HTTP-EQUIV="REFRESH" CONTENT="10;URL=/PUBLIC/TREE.EXE.672137224">
<HEAD>
```

Example:

```
WebWindowClass.CreatePageHeader PROCEDURE(*HtmlClass Target)
```

```
CODE
Target.WriteLine('<<HTML>')
IF (SELF.TimerAction = Update:Refresh)
    Target.WriteRefreshTimer(SELF.TimerDelay)
END
Target.WriteLine('<<HEAD>')
!more page header code
```

## WriteSpace (write HTML space)

---

### WriteSpace(*number*)

**WriteSpace** Writes HTML non-breaking spaces.  
*number* An integer constant, variable, EQUATE, or expression containing the number of spaces to write.

The **WriteSpace** method writes HTML non-breaking spaces. The HtmlClass object uses this method to indent menus.

Implementation: The WriteSpace method writes *number* non-breaking HTML spaces. The HTML symbol for these spaces (across all browsers) is &nbsp;.

Example:

```
WebHtmlMenuClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE  
Target.WriteLine('<<NOBR>')  
Target.WriteSpace((SELF.GetLevel()-1)*2)  
Target.WriteLine(SELF.GetQuotedText())  
Target.WriteLine('<</NOBR><<BR>')
```

## WriteSubmitApplet (write Java applet coordinator)

**WriteSubmitApplet**( [*timer*] [,*action*] )

### WriteSubmitApplet

Writes the HTML code that defines an applet to coordinate communication between the Application Broker and the Java controls on the page.

*timer* An integer constant, variable, EQUATE, or expression containing the milliseconds between browser timer events (the speed of the browser timer). This is typically a function of the value of the `WebWindowClass.TimerDelay` property. If omitted, *timer* defaults to zero (0).

*action* An integer constant, variable, EQUATE, or expression indicating the action to invoke when the *timer* period expires. This is typically the value of the `WebWindowClass.TimerAction` property. If omitted, *action* defaults to zero (0).

The **WriteSubmitApplet** method writes the HTML code that defines an applet to coordinate communication between the Application Broker and the Java controls on the page

Implementation:

The `WriteSubmitApplet` method wrote the following HTML code fragment:

```
<<applet NAME="AppSubmit" CODEBASE="/" CODE=ClarionLoader.class ARCHIVE="/
clarion.zip" MAYSCRIPT WIDTH=2 HEIGHT=2>
<param name=cabbase value="/clarion.cab">
<param name=USID value="/PUBLIC/ORDERS/TREE.EXE.1389494281">
<param name=className value="ClarionCOM">
<param name=UAID value="524289">
<param name=NumListBoxes value="0">
</applet>
```

Example:

```
MyWebWindowClass.CreatePageFooter PROCEDURE(*HtmlClass Target)
CODE
Target.WriteSubmitApplet(SELF.TimerDelay*1000,SELF.TimerAction)
Target.WriteFormFooter
Target.WriteJavaScript
SELF.BodyFooter(Target)
Target.WriteLine('<</BODY>')
Target.WriteLine('<</HTML>')
```

See Also:

`WebWindowClass.TimerAction`, `WebWindowClass.TimerDelay`

## WriteTableFooter (end HTML TABLE)

---

### WriteTableFooter

The **WriteTableFooter** method writes the HTML code to end an HTML TABLE started by the WriteTableHeader method.

Example:

See Also:           WriteTableHeader

## WriteTableHeader (begin HTML TABLE)

---

### WriteTableHeader( [*attr*] )

#### WriteTableHeader

Writes the HTML code to begin an HTML TABLE.

*attr*                   A string constant, variable, EQUATE, or expression containing the attribute to the HTML TABLE. If omitted, the method writes no attribute.

The **WriteTableHeader** method writes the HTML code to begin an HTML TABLE. End the TABLE with the WriteTableFooter method.

Example:

See Also:           WriteTableFooter

## WriteTableNewCol (begin HTML TABLE CELL)

---

### WriteTableNewCol( [*attr*] )

#### WriteTableNewCol

Writes the HTML code to begin an HTML TABLE CELL.

*attr* A string constant, variable, EQUATE, or expression containing the attribute to the HTML TABLE CELL. If omitted, the method writes no attribute.

The **WriteTableNewCol** method writes the HTML code to end the previous TABLE CELL and begin a new CELL. End the CELL with the WriteTableNewCol, WriteTableNewRow, or WriteTableFooter method.

Example:

See Also: WriteTableFooter, WriteTableNewRow

## WriteTableNewRow (begin HTML TABLE ROW)

---

### WriteTableNewRow( [*attr*] )

#### WriteTableNewRow

Writes the HTML code to begin an HTML TABLE ROW.

*attr* A string constant, variable, EQUATE, or expression containing the attribute to the HTML TABLE ROW. If omitted, the method writes no attribute.

The **WriteTableNewRow** method writes the HTML code to end the previous ROW and begin a new ROW. End the ROW with the WriteTableNewRow or WriteTableFooter method.

Example:

See Also: WriteTableFooter

## WriteText (write breakable text string)

---

### WriteText( *text* )

**WriteText**

Writes HTML breakable text string.

*text*

A string constant, variable, EQUATE, or expression containing the text to write, including ASCII carriage return characters to mark the break points.

The **WriteText** method writes HTML breakable text string. The `HtmlClass` object uses this method to indent menus.

Implementation:

The `WriteSpace` method writes HTML to represent *text* by translating ASCII carriage return characters within *text* to their HTML equivalent: `<<BR>`.

Example:

```
WebHtmlStringClass.CreateCellContents PROCEDURE(*HtmlClass Target)
CODE
IF (SELF.CanBreak)
    Target.WriteText(SELF.GetText())
ELSE
    Target.Write('<<NOBR>')
    Target.Write(SELF.GetQuotedText())
    Target.WriteLine('<</NOBR>')
END
```

# ***TEXT OUTPUT CLASS***

<b>Overview</b>	<b>571</b>
TextOutPutClass Concepts .....	571
Relationship to Other Internet Builder Classes .....	571
Internet Connect Template Implementation .....	571
Source Files .....	571
<b>TextOutputClass Methods</b>	<b>572</b>
Functional Organization—Expected Use .....	572
Close (close the file) .....	573
CreateOpen (create and open the file) .....	574
GetSize (return file size) .....	575
Open (open the file) .....	576
Write (write text) .....	577
Writeln (write text and newline marker) .....	578



## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### TextOutPutClass Concepts

---

TextOutPutClass provides basic text file handling, including creating, opening, writing, and closing a text file. This basic file handling is fundamental to the generation of HTML code and Java Support Library (JSL) data by your Web-enabled Clarion application.

Typically your Web-enabled application instantiates a single global TextOutPutClass object.

### Relationship to Other Internet Builder Classes

---

The JslManagerClass instantiates its own TextOutputClass object to write JSL data and protocol that enables fast partial page updates to (just the Java controls on) the generated Web pages.

The HttpClass uses the TextOutputClass to write the HTTP header information that precedes the generated HTML code.

The HtmlClass is derived from the TextOutputClass. The TextOutputClass provides basic file handling methods, including creating, opening, writing, and closing the text file. The HtmlClass adds all the HTML-specific text writing capability.

### Internet Connect Template Implementation

---

The Internet Connect Templates do not directly refer to any TextOutputClass objects.

### Source Files

---

The TextOutputClass source code is installed by default to the \LIBSRC folder. The specific class declarations reside in the following files and their method definitions reside in the corresponding .CLW files.

ICTXTOUT.INC    TextOutputClass

# ***TextOutputClass Methods***

The TextOutputClass contains the methods listed below.

## **Functional Organization—Expected Use**

---

As an aid to understanding the TextOutputClass , it is useful to organize its various methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

### **Primary Interface Methods**

The primary interface methods, which other IBC Library objects call fairly routinely, can be further divided into three categories:

#### **Housekeeping (one-time) Use:**

#### **Mainstream Use:**

CreateOpen	create and open the file
Open	open the file
GetSize	return file size
Write	write text
WriteLn	write text and newline marker
Close	close the file

#### **Occasional Use:**

### **Virtual Methods**

The TextOutputClass has no virtual methods.

## Close (close the file)

---

### Close

The **Close** method closes the text file.

Implementation:       The **Close** method calls the **Writeln** method to write any buffer contents to disk and add a trailing carriage return.

Example:

```
WebWindowClass.CreateHtmlPage PROCEDURE(*HtmlClass Target,STRING HtmlFilename)
CODE
Target.CreateOpen(HtmlFilename,SELF.HtmlOption, |           !open text file
                SELF.Server.JavaClassPath,SELF.Server.Client)
SELF.CreatePageHeader(Target)
SELF.CreateChildHtml(Target,0,SELF.GetTableAttributes())
SELF.CreatePageFooter(Target)
Target.Close           !close text file
SELF.SentHtml = TRUE
```

See Also:               **Writeln**

## CreateOpen (create and open the file)

### CreateOpen( *filename* )

**CreateOpen** Opens the text file.  
*filename* A string constant, variable, EQUATE, or expression containing the pathname of the file to open.

The **CreateOpen** method opens the named text file, creating it first if necessary.

Implementation:

The CreateOpen method creates a file declared in ICTXTOUT.CLW as follows:

```

TargetFilename  CSTRING(FILE:MaxFilepath)
MAXTARGETLEN   EQUATE(8000)

TargetFile  FILE,DRIVER('DOS'),NAME(TargetFilename),PRE(TGT),THREAD,CREATE
RECORD     RECORD,PRE()
TextLine   CSTRING(MAXTARGETLEN+1)
           END
           END

```

Example:

```

MyHttpPageBaseClass.FinishPage PROCEDURE
Protocol          CSTRING(20)
ErrorNum         CSTRING(5)
ErrorMsg         CSTRING(30)

CODE
SELF.FileHandler.CreateOpen(SELF.PageFilename)          !open text file
Protocol = SELF.Http.GetServerProperty('HttpProtocol')
ErrorNum = SELF.Http.GetServerProperty('ErrorNum')
ErrorMsg = SELF.Http.GetServerProperty('ErrorMsg')
SELF.FileHandler.WriteLine(Protocol & ' ' & ErrorNum & ' ' & ErrorMsg)
SELF.WritePageBody()
SELF.FileHandler.WriteLine()
SELF.FileHandler.Close()                               !close text file

```

## GetSize (return file size)

---

### GetSize( *filename* ), LONG

<b>GetSize</b>	Returns the size of the <i>filename</i> file.
<i>filename</i>	A string constant, variable, EQUATE, or expression containing the pathname of the file.

The **GetSize** method returns the size of the *filename* file in bytes. If the file does not exist or cannot be opened, GetSize returns zero (0).

Return Data Type:      **LONG**

Example:

```
HttpPageBaseClass.PreparePage PROCEDURE
CODE
IF (SELF.bHtmlBody)
    SELF.FileLen = SELF.FileHandler.GetSize(SELF.PageFilename) !get file size
    ASSERT(SELF.FileLen<>0)

    IC:RemoveFile(SELF.HtmlFilename)
    IC:RenameFile(SELF.PageFilename, SELF.HtmlFilename)
    SELF.Http.SetServerProperty('Content-length', '' & SELF.FileLen)
ELSE
    SELF.HtmlFilename = SELF.PageFilename
END
SELF.NowDateTime = IC:GetStrDateTime(TODAY(), CLOCK())
SELF.ExpireDateTime = SELF.NowDateTime
SELF.PreparePageBody()
SELF.HandleStatusCode()
```

## Open (open the file)

---

### Open( *filename* )

<b>Open</b>	Opens the text file.
<i>filename</i>	A string constant, variable, EQUATE, or expression containing the pathname of the file to open.

The **Open** method opens the named text file. The file must exist, otherwise the Open method fails.

Implementation: The CreateOpen method offers to GPF (halt) the program if the file does not exist. You (or the end user) may allow the program to continue, but subsequent attempts to access the file will fail.

Example:

```
HttpPageBaseClass.CreatePageBody      PROCEDURE
CODE

SELF.FileHandler.Open(SELF.PageFilename)      !open the text file
SELF.FileHandler.WriteLine('<<HTML>')
SELF.FileHandler.WriteLine('<<P>Error code: ' & SELF.Status & '<</P>')
SELF.FileHandler.WriteLine('<</HTML>')
SELF.FileHandler.Close()                    !close the file
```

## Write (write text)

---

### Write( *text* )

<b>Write</b>	Writes <i>text</i> to the target file.
<i>text</i>	A string constant, variable, EQUATE, or expression containing the text to write.

The **Write** method writes *text* to the target file, without ending the line. Multiple calls to the Write method continue to extend the current line. The WriteLn method ends the current line.

Implementation: The maximum line length is determined by the file definition, typically supplied by the CreateOpen method.

Example:

```
MyWebHtmlStringClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE
Target.Write('<<NOBR>')           !begin line
Target.Write(SELF.GetQuotedText()) !extend line
Target.WriteLn('<</NOBR>')        !end line
```

See Also: CreateOpen, WriteLn

## WriteIn (write text and newline marker)

---

### WriteIn( [*text*] )

<b>WriteIn</b>	Writes <i>text</i> and a newline marker to the target file.
<i>text</i>	A string constant, variable, EQUATE, or expression containing the text to write. If omitted, WriteIn writes only the newline marker.

The **WriteIn** method writes *text* to the target file, ending the current line. The Write method extends the current line.

Implementation: The WriteIn method provides error checking for each write.

Example:

```
MyWebHtmlStringClass.CreateCellContents PROCEDURE(*HtmlClass Target)
```

```
CODE
Target.Write('<<NOBR>')           !begin line
Target.Write(SELF.GetQuotedText()) !extend line
Target.WriteIn('<</NOBR>')        !end line
!end line
```

See Also: Write

# HTTP CLASSES

<b>Overview</b>	<b>581</b>
HttpClass Concepts .....	581
HttpPageBaseClass Concepts .....	581
Relationship to Other Internet Builder Classes .....	582
Internet Connect Template Implementation .....	582
Source Files .....	582
<b>HttpClass Properties</b>	<b>583</b>
Arguments (incoming field data) .....	583
BrowserInfo (Web browser properties) .....	584
Cookies (information stored on Client) .....	585
Files (WebFilesClass object) .....	586
HttpPage (HttpPageBaseClass object) .....	586
ProgName (server name) .....	587
ProcName (password protected Web page) .....	587
ServerInfo (outgoing http information) .....	588
<b>HttpClass Methods</b>	<b>589</b>
Functional Organization—Expected Use .....	589
FinishPage (prepend http to outgoing transmission) .....	591
GetArguments (return incoming field data) .....	591
GetAuthorizedInfo (get client password) .....	592
GetBrowserProperty (return browser property) .....	593
GetCookie (return cookie value) .....	594
GetServerProperty (return outgoing http information) .....	595
Init (initialize the HttpClass object) .....	596
Kill (shut down the HttpClass object) .....	596
PreparePage (prime ServerInfo for next transmission) .....	597
PreparePageForBrowser (prime ServerInfo for HTML transmission) .....	598
PreparePageForJava (prime ServerInfo for JSL transmission) .....	599
PrepareUnauthorized (prime ServerInfo for password challenge) .....	600
ProcessHeader (process incoming http) .....	601
SendCookies (update cookies on client) .....	601
SetBrowserProperty (set browser property) .....	602
SetCookie (set cookie value) .....	603
SetProgName (set server name) .....	604

SetProcName (set protected area name) .....	605
SetServerProperty (set outgoing http item) .....	606
<b>HttpPageBaseClass Properties</b> .....	<b>607</b>
ExpireDateTime (transmission expiration timestamp) .....	607
FileHandler (TextOutputClass object) .....	607
FileLen (transmission size) .....	608
GotHtmlBody (http only transmission) .....	608
HtmlFilename (HTML/JSL transmission file) .....	609
Http (HttpClass object) .....	609
NowDateTime (transmission timestamp) .....	610
PageFilename (entire transmission file) .....	610
Status (status of Client request) .....	611
<b>HttpPageBaseClass Methods</b> .....	<b>612</b>
Functional Organization—Expected Use .....	612
AppendDefaultBody (supply default HTML) .....	613
FinishPage (prepend http to outgoing transmission) .....	614
HandleStatusCode (process status code) .....	615
Init (initialize the HttpPageBaseClass object) .....	616
Kill (shut down the HttpPageBaseClass object) .....	617
PreparePage (set outgoing http items) .....	617
PreparePageBody (virtual to set outgoing http items) .....	618
SetupHttpStatus (set outgoing http status indicators) .....	619
WritePageBody (write outgoing http body) .....	620

## Overview

If you have not already done so, please take a moment to read *Internet Connect Terms and Concepts*. This short topic contains information and terms that are prerequisite to the following material.

### HttpClass Concepts

---

Hyper-Text Transfer Protocol (http) is the protocol that web servers use to transmit HTML pages to internet browsers. Conversely, internet browsers use http to transmit their requests to web servers.

Not surprisingly, then, the IBC Library's `HttpClass` class has two http related jobs:

- To process incoming http headers sent from the Client to the Server. Specifically, it extracts the browser type, any authorization information, and any cookies (Server information stored on the Client) from the incoming http.
- To build http headers for outgoing HTML and Java Support Library (JSL) data, including cookies, demands for authorization, error messages, content size, etc.

### HttpPageBaseClass Concepts

---

The `HttpPageBaseClass` prepares and writes the outgoing http headers for `HttpClass` objects.

The `HttpPageBaseClass` collects the final pieces of http information not provided by the `HttpClass`, then extracts all the outgoing http information in the proper sequence (from the `HttpClass.ServerInfo` property), and finally prepends it to the outgoing HTML or JSL data.

## Relationship to Other Internet Builder Classes

---

The BrokerClass creates and manages a single instance of the HttpClass. The BrokerClass relies on this HttpClass object to do both jobs described above. That is, to process incoming http headers, and to build http headers for outgoing HTML and JSL data.

The HttpClass is derived from the HttpBaseClass class. The HttpBaseClass class is an abstract class whose methods are not defined. It provides a reference for all of its derived classes, including the HttpClass.

The HttpClass relies on the WebFilesClass to provide appropriate filenames and pathnames. It also relies on the TextOutputClass to provide basic file handling, including creating, opening, writing to, and closing its files.

## Internet Connect Template Implementation

---

The BrokerClass creates and manages a single instance of the HttpClass. Therefore the template generated code does not directly reference the HttpClass object.

## Source Files

---

The HttpClass source code is installed by default to the \LIBSRC folder. The specific class declarations are in the following files and their method definitions are in the corresponding .CLW files.

ICHTTP.INC	HttpClass
------------	-----------

## *HttpClass Properties*

The `HttpClass` contains the properties listed below.

### Arguments (incoming field data)

---

Arguments	ANY, PROTECTED
	<p>The <b>Arguments</b> property contains data entered through the Client's Web page. The <code>GetArguments</code> method returns the value of the <code>Arguments</code> property.</p> <p>This property is <code>PROTECTED</code>, therefore, it can only be referenced by an <code>HttpClass</code> method, or a method in a class derived from <code>HttpClass</code>.</p>
Implementation:	<p>The <code>Arguments</code> property contains a list of field assignments of the form 'field1=value1&amp;field2=value2&amp;fieldN=valueN'. The <code>SubmitItemClass</code> manages a single field assignment within this string.</p> <p>The <code>ProcessHeader</code> method sets the value of the <code>Arguments</code> property based on the incoming http which contains something like the following http fragment:</p> <pre>GET /PUBLIC/TREE.EXE.1043595275?X30003=0H&amp;X30005=0Hio&amp;X30006=0K HTTP/1.0</pre>
See Also:	<code>GetArguments</code> , <code>ProcessHeader</code> , <code>SubmitItemClass</code>

## BrowserInfo (Web browser properties)

---

### BrowserInfo    &HttpInfoQueue

The **BrowserInfo** property is a reference to a structure that contains information about the Client's Web browser.

The SetBrowserProperty method adds BrowserInfo items. The GetBrowserProperty method returns specific BrowserInfo items.

#### Implementation:

The BrowserInfo property is inherited from the HttpBaseClass class. It contains information extracted from the incoming http header by the ProcessHeader method. The information is stored in a queue with the same structure as the HttpInfoQueue declared in ICHTTP.INC as follows:

```
HttpInfoQueue        QUEUE,TYPE
Name                CSTRING(40)
Value                CSTRING(255)
Special              SIGNED
                      END
```

#### Typical BrowserInfo items include:

```
User-Agent: Mozilla/2.0 (compatible; MSIE 3.01; Windows 95)
Authorization: Basic 0nBhc3M=
Extension: Security/Remote-Passphrase
Accept-Language: en
UA-pixels: 800x600
UA-color: color8
UA-OS: Windows 95
UA-CPU: x86
```

#### See Also:

GetBrowserProperty, ProcessHeader, SetBrowserProperty

## Cookies (information stored on Client)

### Cookies&CookiesQueue

The **Cookies** property is a reference to a structure that contains information stored on the Client machine (cookies) at the request of the Server.

A server can send data to a client which the client stores locally. This is known as a cookie. A cookie contains a range of URLs for which it is valid. When the client returns to a URL within that range, the server can query the cookie and use its data. A server cannot query a cookie from another server.

This mechanism provides a way to maintain persistent user-specific information between remote computing sessions. For example, an application which requires a user to provide a username can use a cookie to avoid the Login process after the first visit. Your web-enabled applications can use cookies to store user preferences such as the default city and state for new records. These settings can be retrieved each time the user runs the application over the web.

Cookies are machine and browser specific so a client who accesses a site from more than one machine or uses more than one browser will need to provide the cookie information once for each machine and browser.

The `GetCookie` method returns a specific cookie item from the Client. The `SetCookie` method sets a specific cookie item to send to the Client. The `SendCookies` method updates the cookies on the Client machine.

#### Implementation:

The `Cookies` property is inherited from the `HttpBaseClass` class. It contains information (cookies) extracted from the incoming http header by the `ProcessHeader` method. The information is stored in a queue with the same structure as the `CookiesQueue` declared in `ICHTTP.INC` as follows:

```

CookiesQueue      QUEUE, TYPE
Name              CSTRING(40)
Value             CSTRING(255)
Path              CSTRING(255)
ExpireDate        LONG
ExpireTime        LONG
Modified          BYTE
                  END

```

#### See Also:

`GetCookie`, `ProcessHeader`, `SendCookies`, `SetCookie`

## Files (WebFilesClass object)

---

### Files      &WebFilesClass

The **Files** property is a reference to the WebFilesClass object that provides all the filenames, pathnames, directories, and aliases required by the HttpClass object.

Implementation:      The Init method sets the value of the Files property.

See Also:              Init

## HttpPage (HttpPageBaseClass object)

---

### HttpPage      &HttpPageBaseClass, PROTECTED

The **HttpPage** property is a reference to the HttpPageBaseClass object that prepares and writes the outgoing http headers for the HttpClass object.

This property is PROTECTED, therefore, it can only be referenced by an HttpClass method, or a method in a class derived from HttpClass.

Implementation:      The PreparePage method sets the value of the HttpPage property. The HttpPage property inputs the final pieces of information into the ServerInfo property, then extracts all the outgoing http information in the proper sequence from the ServerInfo property, and prepends it to the outgoing HTML or JSL data.

See Also:              HttpPageBaseClass, PreparePage, ServerInfo

## ProgName (server name)

---

<b>ProgName</b>	<b>CSTRING(255), PROTECTED</b>
-----------------	--------------------------------

The **ProgName** property identifies the Server. Typically this is the name of the program as launched by the Application Broker; however, it may be any value that appropriately identifies the Server.

The SetProgName method sets the value of this property.

This property is PROTECTED, therefore, it can only be referenced by an HttpClass method, or a method in a class derived from HttpClass.

Implementation: The HttpClass object uses the ProgName property to identify the Server for password protected areas and for cookies.

See Also: Cookies, SetProgName, PrepareUnauthorized

## ProcName (password protected Web page)

---

<b>ProcName</b>	<b>CSTRING(255), PROTECTED</b>
-----------------	--------------------------------

The **ProcName** property names a Web page that is password protected. Typically this is the name of the procedure that generates the protected Web page; however, it may be any value that appropriately identifies the protected area to end users.

The SetProcName method sets the value of this property. The PrepareUnauthorized method prepares the password challenge to send to the Client.

This property is PROTECTED, therefore, it can only be referenced by an HttpClass method, or a method in a class derived from HttpClass.

See Also: SetProcName, PrepareUnauthorized

## ServerInfo (outgoing http information)

### ServerInfo &HttpInfoQueue

The **ServerInfo** property is a reference to a structure that contains information about the Server transmission to the Client. This information is passed to the Client within the outgoing http header generated by the `HttpClass` object.

The `GetServerProperty` method returns a specific `ServerInfo` item. The `SetServerProperty` method sets a specific `ServerInfo` item.

#### Implementation:

The `ServerInfo` property is inherited from the `HttpBaseClass` class. This property contains information set by the `HttpClass` object. The information is stored in a queue with the same structure as the `HttpInfoQueue` declared in `ICHTTP.INC` as follows:

```

HttpInfoQueue    QUEUE,TYPE
Name             CSTRING(40)
Value           CSTRING(255)
Special         SIGNED
                END

```

The `HttpClass` object adds information to the `ServerInfo` queue as the information becomes available. Then, the `HttpClass` object uses the `HttpPage` property (an `HttpPageBaseClass` object) to extract the information from the queue in the sequence needed to create the http header. The `HttpClass` object frees and reallocates the `ServerInfo` queue between page transmissions.

Typical `ServerInfo` items include:

```

HttpProtocol: HTTP/1.0
Server:
Content-length:
Content-type:
Last-modified:
Date:
Expires:
Allowed:
MIME-Version:
WWW-authenticate:
Set-Cookie:
Pragma: no-cache
ErrorNum:
ErrorMsg:

```

#### See Also:

`GetServerProperty`, `SetServerProperty`

## HttpClass Methods

The HttpClass contains the methods listed below.

### Functional Organization—Expected Use

---

As an aid to understanding the HttpClass, it is useful to organize its various methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

#### Primary Interface Methods

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

##### **Housekeeping (one-time) Use:**

Init	initialize the HttpClass object
Kill	shut down the HttpClass object

##### **Mainstream Use:**

GetCookie	return cookie value
SetCookie	set cookie value
SetProcName	set protected area name
SetProgName	set server name

##### **Occasional Use:**

ProcessHeader	process incoming http
GetArguments	return incoming field data
GetAuthorizedInfo	get Client password
PreparePageForBrowserprime	ServerInfo for HTML page
PreparePageForJava prime	ServerInfo for JSL page
PrepareUnauthorized prime	ServerInfo for password page
FinishPage	prepend http to outgoing page

## **Virtual Methods**

Typically you will not call these methods directly—the Primary Interface methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

GetBrowserProperty	return browser property
GetServerProperty	return outgoing http information
SetBrowserProperty	set browser property
SetServerProperty	set outgoing http information
PreparePage	prime ServerInfo for next page
SendCookies	update cookies on client

## FinishPage (prepend http to outgoing transmission)

---

### FinishPage

The **FinishPage** method writes the http text appropriate to the outgoing transmission (HTML code or JSL data) then prepends it to the outgoing transmission. Finally, FinishPage prepares the HttpClass object to handle the next transmission.

Implementation: The FinishPage method extracts http information from the ServerInfo property, formats it to HTTP/1.0 standards, then prepends it to the outgoing transmission.

The FinishPage method prepares for the next transmission by DISPOSEing the HttpPage property and the ServerInfo property.

Example:

```
BrokerClass.TakeHtmlPage PROCEDURE(String Filename,SIGNED Security,BYTE dontmove)
CODE
SELF.Http.PreparePageForBrowser(200, Filename)
SELF.Http.FinishPage()
SELF.TakeFile(Filename,Security,dontmove)
```

See Also: [HttpPage](#), [ServerInfo](#)

## GetArguments (return incoming field data)

---

### GetArguments, STRING

The **GetArguments** method returns the value of the Arguments property.

Return Data Type: **STRING**

Example:

```
BrokerClass.GetRequestArguments PROCEDURE
CODE
RETURN SELF.Http.GetArguments()
```

See Also: [Arguments](#)

## GetAuthorizedInfo (get client password)

---

**GetAuthorizedInfo**( *username, password* )

### **GetAuthorizedInfo**

Gets the Client's (end user) username and password.

*username*            A string variable to receive the Client's username.

*password*           A string variable to receive the Client's password.

The **GetAuthorizedInfo** method gets the Client's (end user) username and password.

Implementation:        The **GetAuthorizedInfo** method calls the **GetBrowserProperty** method to get the information from the **BrowserInfo** property.

Example:

```
BrokerClass.GetAuthorizedInfo PROCEDURE(STRING AreaName,*STRING User,*STRING Password)
CODE
SELF.Http.SetProcName(AreaName)
SELF.Http.GetAuthorizedInfo(User, Password)
```

See Also:                **BrowserInfo, GetBrowserProperty**

## GetBrowserProperty (return browser property)

---

**GetBrowserProperty( *property* ), STRING, VIRTUAL**

### GetBrowserProperty

Returns the specified browser property.

*property* A string constant, variable, EQUATE, or expression identifying the property to return.

The **GetBrowserProperty** method returns a browser property that was set by the **SetBrowserProperty** method.

**GetBrowserProperty** is a **VIRTUAL** method so that other base class methods can directly call the **GetBrowserProperty** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **GetBrowserProperty** method gets the information from the **BrowserInfo** property.

Return Data Type: **STRING**

Example:

```
BrokerClass.SetClientBrowser PROCEDURE
BrowserType      STRING(100)
CODE
BrowserType = SELF.Http.GetBrowserProperty('User-Agent')
IF (INSTRING('MSIE 4.', BrowserType, 1, 1))
  SELF.CurClient.Browser &= IE40
ELSIF (INSTRING('MSIE 3.', BrowserType, 1, 1))
  SELF.CurClient.Browser &= IE30
ELSIF (INSTRING('Mozilla/4.', BrowserType, 1, 1))
  SELF.CurClient.Browser &= Mozilla4
ELSIF (INSTRING('Mozilla/3.', BrowserType, 1, 1))
  SELF.CurClient.Browser &= NetScape3x
END
```

See Also: **BrowserInfo, SetBrowserProperty**

## GetCookie (return cookie value)

---

### **GetCookie( *cookie* ), STRING**

**GetCookie** Returns the specified cookie value.  
*cookie* A string constant, variable, EQUATE, or expression identifying the cookie whose value to return.

The **GetCookie** method returns the specified cookie value.

Implementation: The GetCookie method gets the information from the Cookies property. The Cookies property may contain cookie values that came from the Client (set by the ProcessHeader method) as well as cookie values set by the Server (set by the SetCookie method) to send to the Client.

Return Data Type: **STRING**

Example:

```
PrepareProcedure ROUTINE
  DefaultCity = Broker.Http.GetCookie('City')

ProcedureReturn ROUTINE
  Broker.Http.SetCookie('City', DefaultCity)
```

See Also: **Cookies, SetCookie**

## GetServerProperty (return outgoing http information)

**GetServerProperty( *item* ), STRING, VIRTUAL**

### GetServerProperty

Returns the specified outgoing http information.

*item* A string constant, variable, EQUATE, or expression identifying the item to return.

The **GetServerProperty** method returns an information item that was set by the **SetServerProperty** method. The returned information is typically used to build the outgoing http headers.

**GetServerProperty** is a **VIRTUAL** method so that other base class methods can directly call the **GetServerProperty** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **GetServerProperty** method gets the information from the **ServerInfo** property.

Return Data Type: **STRING**

Example:

```
HttpPageBaseClass.FinishPage PROCEDURE
Protocol          CSTRING(20)
ErrorNum          CSTRING(5)
ErrorMsg          CSTRING(30)

CODE
SELF.FileHandler.CreateOpen(SELF.PageFilename)

Protocol = SELF.Http.GetServerProperty('HttpProtocol')
ErrorNum = SELF.Http.GetServerProperty('ErrorNum')
ErrorMsg = SELF.Http.GetServerProperty('ErrorMsg')

SELF.FileHandler.WriteLine(Protocol & ' ' & ErrorNum & ' ' & ErrorMsg)
SELF.WritePageBody()
SELF.FileHandler.WriteLine()
SELF.FileHandler.Close()
IF (SELF.GotHtmlBody)
    SELF.AppendFileTo(SELF.HtmlFilename, SELF.PageFilename, SELF.FileLen, TRUE)
ELSE
    SELF.AppendDefaultBody()
END
```

See Also: **ServerInfo, SetServerProperty**

## Init (initialize the HttpClass object)

---

### Init( *files* )

**Init** Initializes the HttpClass object.

*files* The label of the WebFilesClass object that provides all the filenames, pathnames, directories, and aliases required by the HttpClass object.

The **Init** method initializes the HttpClass object.

Implementation: The Init method sets the value of the Files property. It also instantiates the BrowserInfo, Cookies, and ServerInfo properties.

Example:

```
MyBrokerClass.Init PROCEDURE(STRING ProgramName, WebFilesClass Files)
CODE
SELF.Files &= Files
SELF.Http &= NEW HttpClass           !instantiate new HttpClass object
SELF.Http.Init(Files)                 !and initialize it
SELF.Http.SetProgName(ProgramName)
```

## Kill (shut down the HttpClass object)

---

### Kill

The **Kill** method frees all memory allocated during the life of the object and performs any other required termination code.

Implementation: The Kill method DISPOSEs the BrowserInfo, Cookies, and ServerInfo properties.

Example:

```
MyBrokerClass.Kill PROCEDURE
CODE
SELF.CloseChannel
IF (NOT (SELF.Http &= NULL))
    SELF.Http.Kill           !shut down HttpClass object
    DISPOSE(SELF.Http)
END
```

## PreparePage (prime ServerInfo for next transmission)

**PreparePage( *http page*, *status* [,*filename*] ), VIRTUAL, PROTECTED**

<b>PreparePage</b>	Primes the ServerInfo property with http information pertaining to the next outgoing transmission.
<i>http page</i>	The label of the HttpPageBaseClass object that prepares and writes the outgoing http header for the HttpClass object.
<i>status</i>	An integer constant, variable, EQUATE, or expression containing the status code to send to the Client. The status code indicates the status of the Server's response to the Client's request.
<i>filename</i>	A string constant, variable, EQUATE, or expression containing the name of the file to send to the Client browser. If omitted, the HttpClass object creates a dummy page to send.

The **PreparePage** method primes the ServerInfo property with http information pertaining to the next outgoing transmission (HTML or JSL data).

PreparePage is a VIRTUAL method so that other base class methods can directly call the PreparePage virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method is PROTECTED, therefore, it can only be called by an HttpClass method, or a method in a class derived from HttpClass.

Implementation:

The PreparePage method prepares for the transmission by DISPOSEing the HttpPage property and the ServerInfo property. Then it calls the HttpPageBaseClass.PreparePage method to prime the ServerInfo property with http settings for the next transmission.

Typically, the *filename* parameter names the file containing HTML code or JSL data prepared by the WebWindow class object. The BrokerClass object passes the filename to the HttpClass object, which passes it on to the HttpPageBaseClass object.

Example:

```
HttpClass.PreparePageForBrowser PROCEDURE(SIGNED status, <STRING Filename>)
HttpPage &HttpPageBaseClass
```

```
CODE
HttpPage &= NEW HttpPageBrowserClass
SELF.PreparePage(HttpPage, status, Filename)
```

See Also:

HttpPage, HttpPageBaseClass.PreparePage, HttpPageBaseClass..Status, ServerInfo

## PreparePageForBrowser (prime ServerInfo for HTML transmission)

**PreparePageForBrowser**( *status* [, *filename*] )

### **PreparePageForBrowser**

Primes the ServerInfo property with http information pertaining to the next outgoing HTML transmission.

*status* An integer constant, variable, EQUATE, or expression containing the status code to send to the Client. The status codes indicates the Server's response to the Client's request.

*filename* A string constant, variable, EQUATE, or expression containing the name of the file to send to the Client. If omitted, ...

The **PreparePageForBrowser** method primes the ServerInfo property with http information pertaining to the next outgoing HTML transmission.

Implementation: The PreparePageForBrowser method calls the PreparePage method to prime the ServerInfo property with http settings for the transmission.

Example:

```

HttpClass.PrepareUnauthorized PROCEDURE(<STRING Filename>)
HttpPage &HttpPageBaseClass
CODE
SELF.PreparePageForBrowser(401, Filename)           !prepare password challenge page
SELF.SetServerProperty('WWW-authenticate','basic realm="'&SELF.ProcName&'")

BrokerClass.TakeHtmlPagePROCEDURE(STRING Filename, SIGNED Security, BYTE dontmove)
CODE
SELF.Http.PreparePageForBrowser(200, Filename)       !prepare new html page
IF (SELF.CurClient.Browser.SetNoCache)
    SELF.Http.SetServerProperty('Pragma', 'no-cache')
END
SELF.Http.FinishPage()
SELF.TakeFile(Filename, Security, dontmove)

```

See Also: **PreparePage**, **HttpPageBaseClass.Status**, **ServerInfo**

## PreparePageForJava (prime ServerInfo for JSL transmission)

---

**PreparePageForJava( *status* [,*filename*] )**

### **PreparePageForJava**

Primes the ServerInfo property with http information pertaining to the next outgoing JSL data transmission.

*status* An integer constant, variable, EQUATE, or expression containing the status code to send to the Client. The status codes indicates the Server's response to the Client's request.

*filename* A string constant, variable, EQUATE, or expression containing the name of the file to send to the Client. If omitted, ...

The **PreparePageForJava** method primes the ServerInfo property with http information pertaining to the next outgoing JSL data transmission.

Implementation: The PreparePageForJava method calls the PreparePage method to prime the ServerInfo property with http settings for the transmission.

Example:

```
BrokerClass.TakeJsldata PROCEDURE(STRING Filename, SIGNED Security)
CODE
SELF.Http.PreparePageForJava(200, Filename)
SELF.Http.FinishPage()
SELF.TakeFile(Filename, Security, TRUE)
```

See Also: PreparePage, HttpPageBaseClass.Status, ServerInfo

## PrepareUnauthorized (prime ServerInfo for password challenge)

---

### PrepareUnauthorized( *filename* )

#### **PrepareUnauthorized**

Primes the ServerInfo property with http information pertaining to a password protected area.

*filename* A string constant, variable, EQUATE, or expression containing the name of the file to send to the Client. If omitted, ...

The **PrepareUnauthorized** method primes the ServerInfo property with http information pertaining to a password protected HTML page.

Implementation: The PrepareUnauthorized method calls the PreparePage method to prime the ServerInfo property with http settings for the transmission.

Example:

```
BrokerClass.TakeUnauthorized PROCEDURE(STRING Filename, SIGNED Security)
CODE
SELF.Http.PrepareUnauthorized(Filename)
IF (SELF.CurClient.Browser.SetNoCache)
    SELF.Http.SetServerProperty('Pragma', 'no-cache')
END
SELF.Http.FinishPage()
SELF.TakeFile(Filename, Security, TRUE)
```

See Also: **PreparePage, ServerInfo**

## ProcessHeader (process incoming http)

---

### ProcessHeader( *http text* )

**ProcessHeader** Processes incoming http headers.

*http text* A string constant, variable, EQUATE, or expression containing the http text to process.

The **ProcessHeader** method processes incoming http headers. It extracts information for the **BrowserInfo** property, the **Cookies** property, and the **Arguments** property.

Implementation: The **ProcessHeader** method clears and reloads the **Arguments**, **BrowserInfo**, and **Cookies** properties.

Example:

```
BrokerClass.ProcessHttpHeader PROCEDURE(STRING HeaderText)
CODE
SELF.Http.ProcessHeader(HeaderText)
```

See Also: Arguments, BrowserInfo, Cookies

## SendCookies (update cookies on client)

---

### SendCookies, VIRTUAL

The **SendCookies** method primes the **ServerInfo** property with cookie information for the next outgoing transmission.

**SendCookies** is a **VIRTUAL** method so that other base class methods can directly call the **SendCookies** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **SendCookies** method extracts current cookie information from the **Cookies** property, and calls the **SetServerProperty** method to add the cookie information to the **ServerInfo** property. The **FinishPage** method includes the **ServerInfo** information (including the cookies) in the outgoing http.

Example:

```
HttpPageBrowserClass.FinishPage PROCEDURE
CODE
SELF.Http.SendCookies()
PARENT.FinishPage()
```

See Also: Cookies, GetCookie, ServerInfo, SetCookie

## SetBrowserProperty (set browser property)

---

**SetBrowserProperty**( *property*, *value* ), VIRTUAL, PROTECTED

### **SetBrowserProperty**

Sets the value of the specified browser property.

*property* A string constant, variable, EQUATE, or expression identifying the property whose *value* to set.

*value* A string constant, variable, EQUATE, or expression containing the contents of the *property*.

The **SetBrowserProperty** method sets the value of the specified browser property. The **GetBrowserProperty** method returns values set by the **SetBrowserProperty** method.

**SetBrowserProperty** is a VIRTUAL method so that other base class methods can directly call the **SetBrowserProperty** virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method is PROTECTED, therefore, it can only be called by an **HttpClass** method, or a method in a class derived from **HttpClass**.

Implementation: The **SetBrowserProperty** method adds items to the **BrowserInfo** property.

Example:

```
SELF.SetBrowserProperty('HttpProtocol', 'HTTP/1.0')
```

See Also: **BrowserInfo**, **GetBrowserProperty**

## SetCookie (set cookie value)

**SetCookie**( *cookie*, *value* [,*expire date*] [,*expire time*] [,*path*] )

<b>SetCookie</b>	Sets the value of the specified cookie.
<i>cookie</i>	A string constant, variable, EQUATE, or expression identifying the cookie whose <i>value</i> to set.
<i>value</i>	A string constant, variable, EQUATE, or expression containing the contents of the <i>cookie</i> . If <i>value</i> is set to null (''), the Client's browser deletes the cookie.
<i>expire date</i>	An integer constant, variable, EQUATE, or expression containing the cookie expiration date. If omitted, <i>expire date</i> and <i>expire time</i> is set to 28 days from the present system time.
<i>expire time</i>	An integer constant, variable, EQUATE, or expression containing the cookie expiration time. If omitted, <i>expire time</i> defaults to zero (0).
<i>path</i>	A string constant, variable, EQUATE, or expression containing the path of the cookie. If omitted, the path defaults to null (''). The Client's browser determines how the path information is applied.

The **SetCookie** method sets the value of the specified cookie. When the Client receives the cookie information, if *value* is set to null (''), the Client's browser deletes the cookie.

The **GetCookie** method returns cookie values set by the **SetCookie** method or by the **ProcessHeader** method.

**Implementation:** The **SetCookie** method updates the **Cookie** property. The **FinishPage** method includes the contents of the **Cookie** property in the outgoing http header.

**Example:**

```
PrepareProcedure ROUTINE
    DefaultCity = Broker.Http.GetCookie('City')

ProcedureReturn ROUTINE
    Broker.Http.SetCookie('City', DefaultCity)
```

**See Also:** [Cookies](#), [FinishPage](#), [GetCookie](#)

## SetProgName (set server name)

---

### SetProgName( *name* )

<b>SetProgName</b>	Sets the Server name.
<i>name</i>	A string constant, variable, EQUATE, or expression identifying the server.

The **SetProgName** method sets the Server name.

The *name* parameter typically contains the name of the program as launched by the Application Broker; however, it may contain any value that appropriately identifies the Server.

Implementation: The SetProgName method sets the value of the ProgName property. The HttpClass object uses the ProgName property to identify the Server for password protected areas and for cookies.

Example:

```
Broker.Http.SetProgName('Net Orders')
```

See Also: Cookies, ProgName, PrepareUnauthorized

## SetProcName (set protected area name)

---

### SetProcName( *name* )

**SetProcName** Sets the name of a password protected Web page.  
*name* A string constant, variable, EQUATE, or expression identifying the password protected Web page.

The **SetProcName** method sets the name of a password protected Web page (or area, if the protected page is the entry point to other pages).

The *name* parameter typically contains the name of the procedure that generates the protected Web page; however, it may contain any value that appropriately identifies the password protected area to end users, because this is the name end users see when the Client browser prompts for the password.

Implementation: The **SetProcName** method sets the value of the ProcName property. The HttpClass object uses the both the ProgName and ProcName properties to identify password protected Web pages.

Example:

```
BrokerClass.GetAuthorizedInfo PROCEDURE(STRING AreaName,*STRING User,*STRING Password)
CODE
SELF.Http.SetProcName(AreaName)
SELF.Http.GetAuthorizedInfo(User, Password)
```

The **PrepareUnauthorized** method prepares the password challenge to send to the Client.

See Also: ProcName, ProgName, PrepareUnauthorized

## SetServerProperty (set outgoing http item)

**SetServerProperty( *name*, *value* [,*special*] [,*multiple*] ), VIRTUAL**

### SetServerProperty

Sets the value of the specified outgoing http item.

*name* A string constant, variable, EQUATE, or expression identifying the property whose *value* to set.

*value* A string constant, variable, EQUATE, or expression containing the contents of the item.

*special* An integer constant, variable, EQUATE, or expression indicating whether the item requires special handling. If omitted, *special* defaults to zero(0).

*multiple* An integer constant, variable, EQUATE, or expression indicating whether the item replaces other items with the same *name*. A value of one(1) allows multiple items with the same *name*; a value of zero(0) allows only one item with the same *name*. If omitted, *multiple* defaults to zero(0). For example, there may be more than one “Set-Cookie” item, but there is only one “Server” item.

The **SetServerProperty** method sets the value of the specified outgoing http item. The **GetServerProperty** method returns values set by the **SetServerProperty** method.

**SetServerProperty** is a VIRTUAL method so that other base class methods can directly call the **SetServerProperty** virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: The **SetServerProperty** method adds items to the **ServerInfo** property. The **FinishPage** method includes these items in the outgoing http header.

Example:

```
HttpClass.SendCookies    PROCEDURE
NumCookies              SIGNED,AUTO
ExtendedValue          CSTRING(1000)
DateTime               CSTRING(100)
I                      SIGNED,AUTO
CODE
NumCookies = RECORDS(SELF.Cookies)
LOOP I = 1 TO NumCookies
  GET(SELF.Cookies, I)
  ExtendedValue = SELF.Cookies.Name&' '&SELF.Cookies.Value&' '
  SELF.SetServerProperty('Set-Cookie', ExtendedValue, FALSE, TRUE)
END
```

See Also: **ServerInfo**, **FinishPage**, **GetServerProperty**

## *HttpPageBaseClass Properties*

The `HttpPageBaseClass` contains the properties listed below.

### **ExpireDateTime (transmission expiration timestamp)**

---

#### **ExpireDateTime STRING(100), PROTECTED**

The **ExpireDateTime** property contains the date and time the transmitted page expires or is no longer valid.

This property is PROTECTED, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `PreparePage` method sets the value of the `ExpireDateTime` property. The `HttpClass` object includes this timestamp in the `http` header it sends to the Client browser. The browser determines how the timestamp is actually used.

See Also: `PreparePage`

### **FileHandler (TextOutputClass object)**

---

#### **FileHandler &TextOutputClass, PROTECTED**

The **FileHandler** property is a reference to the `TextOutputClass` object that handles basic file handling, such as creating, opening, writing, and closing, for the `HttpPageBaseClass` object.

This property is PROTECTED, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `Init` method instantiates a new `FileHandler` for the `HttpPageBaseClass` object.

See Also: `Init`, `TextOutputClass`

## FileLen (transmission size)

---

### FileLen ULONG, PROTECTED

The **FileLen** property contains the size of the entire transmission, including the http header information. The `HttpPageBaseClass` includes the transmission size in the outgoing http header.

This property is **PROTECTED**, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `PreparePage` method sets the value of the `FileLen` property.

See Also: `PreparePage`

## GotHtmlBody (http only transmission)

---

### GotHtmlBody BYTE, PROTECTED

The **GotHtmlBody** property indicates whether a body file (containing HTML code or JSL data) was passed to the `HttpClass` object. A value of one (1) indicates a body file was passed; a value of zero (0) indicates no body was passed and the `HttpClass` object should supply a default HTML body.

This property is **PROTECTED**, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `Init` method sets the value of the `GotHtmlBody` property.

See Also: `Init`

## HtmlFilename (HTML/JSL transmission file)

---

**HtmlFilename** CSTRING(FILE:MaxFileName), PROTECTED

The **HtmlFilename** property identifies the temporary file that contains only the body of the transmission—that is, the file that contains the HTML code or JSL data, but does not contain the http header information.

This property is PROTECTED, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `Init` method sets the value of the `HtmlFilename` property.

See Also: `Init`

## Http (HttpClass object)

---

**Http** &`HttpBaseClass`, PROTECTED

The **Http** property is a reference to the (parent) `HttpClass` object that handles both incoming and outgoing http headers.

This property is PROTECTED, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `Init` method sets the value of the `Http` property. The `HttpPageBaseClass` object uses the `Http` property to access the http information (`ServerInfo`) already collected by and resident in the `Http` property. The `HttpPageBaseClass` object also uses some of the `Http` property's methods (`SetServerProperty`) to add the final pieces of http information before assembling the http header.

See Also: `HttpClass.SetServerProperty`, `HttpClass.ServerInfo`, `Init`

## NowDateTime (transmission timestamp)

---

**NowDateTime** STRING(100), PROTECTED

The **NowDateTime** property contains the date and time on the server machine when the transmitted file was created. The `HttpPageBaseClass` object uses this property to calculate the `ExpireDateTime` property.

This property is `PROTECTED`, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `PreparePage` method sets the value of the `NowDateTime` property.

See Also: `PreparePage`, `ExpireDateTime`

## PageFilename (entire transmission file)

---

**PageFilename** CSTRING(FILE:MaxFileName), PROTECTED

The **PageFilename** property identifies the file output by the `HttpPageBaseClass` object that contains the entire transmission—that is, the file that ultimately contains both the http header and the HTML code or JSL data.

This property is `PROTECTED`, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `Init` method sets the value of the `PageFilename` property.

See Also: `Init`

## Status (status of Client request)

---

### Status SIGNED, PROTECTED

The **Status** property contains the status code the Server returns to the Client. The Status property indicates the status of the Client request.

This property is PROTECTED, therefore, it can only be referenced by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

#### Implementation:

The `Init` method sets the value of the Status property. The `HandleStatusCode` and `SetupHttpStatus` methods set a corresponding message. Valid status codes and messages include:

Code	Message
200	'OK'
201	'Created'
202	'Accepted'
204	'No Content'
300	'Multiple Choices'
301	'Moved Permanently'
302	'Moved Temporarily'
304	'Not Modified'
400	'Bad Request'
401	'Unauthorized'
403	'Forbidden'
404	'Not Found'
500	'Internal Server Error'
501	'Not Implemented'
502	'Bad Gateway'
503	'Service Unavailable'
other	'UNKNOWN!'

#### See Also:

`HandleStatusCode`, `Init`, `SetupHttpStatus`

# HttpPageBaseClass Methods

The HttpPageBaseClass contains the methods listed below.

## Functional Organization—Expected Use

---

As an aid to understanding the HttpPageBaseClass, it is useful to organize its methods into two categories according to their expected use—the primary interface and the virtual methods. This organization reflects what we believe is typical use of these methods.

### Primary Interface Methods

The primary interface methods, which you are likely to call fairly routinely from your program, can be further divided into three categories:

#### **Housekeeping (one-time) Use:**

Init	initialize HttpPageBaseClass object
Kill	shut down HttpPageBaseClass object

#### **Mainstream Use:**

#### **Occasional Use:**

SetupHttpStatus	set outgoing http status indicators
-----------------	-------------------------------------

### Virtual Methods

Typically you will not call these methods directly—the Primary Interface methods call them. However, we anticipate you will often want to override these methods, and because they are virtual, they are very easy to override. These methods do provide reasonable default behavior in case you do not want to override them.

AppendDefaultBody	supply default HTML
FinishPage	prepend http to outgoing transmission
HandleStatusCode	process status code
PreparePage	set outgoing http items
PreparePageBody	virtual to set outgoing http items
WritePageBody	write outgoing http body

## AppendDefaultBody (supply default HTML)

---

### AppendDefaultBody, VIRTUAL, PROTECTED

The **AppendDefaultBody** method creates and appends some default HTML code to the transmission. The `HttpPageBaseClass` uses this method to supply some meaningful HTML when the Client requested a new HTML page, but the Server was otherwise unable to generate it.

`AppendDefaultBody` is a VIRTUAL method so that other base class methods can directly call the `AppendDefaultBody` virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method is PROTECTED, therefore, it can only be called by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation: The `AppendDefaultBody` method creates HTML to display the status code (Status property) for the Client's request. The `GotHtmlBody` property indicates whether the expected HTML is present or absent.

Example:

```
HttpPageBaseClass.FinishPage  PROCEDURE
CODE
!procedure code

IF (SELF.GotHtmlBody)                !if HTML exists, merge it
    SELF.AppendFileTo(SELF.HtmlFilename,SELF.PageFilename,SELF.FileLen,TRUE)
ELSE                                  !if HTML is missing
    SELF.AppendDefaultBody()         !create some to display the status code
END
```

See Also: `GotHtmlBody`, `Status`

## FinishPage (prepend http to outgoing transmission)

---

### FinishPage, VIRTUAL

The **FinishPage** method writes the http text appropriate to the outgoing transmission (HTML code or JSL data) then prepends it to the outgoing transmission.

FinishPage is a VIRTUAL method so that other base class methods can directly call the FinishPage virtual method in a derived class. This lets you easily implement your own custom version of this method.

#### Implementation:

The `HttpClass.FinishPage` method calls the `httppagebaseclassderived.FinishPage` method. The `FinishPage` method extracts http information from the `ServerInfo` property, formats it to HTTP/1.0 standards, then prepends it to the outgoing transmission.

#### Example:

```
HttpPageBrowserClass.FinishPage PROCEDURE           !derived class FinishPage method
CODE
SELF.Http.SendCookies()
PARENT.FinishPage()                               !calls HttpPageBaseClass.FinishPage
```

#### See Also:

`HttpClass.FinishPage`, `HttpPageBrowserClass.FinishPage`

## HandleStatusCode (process status code)

---

### HandleStatusCode, VIRTUAL, PROTECTED

The **HandleStatusCode** method performs all required status code processing, including setting a corresponding status message, and any other http items that are status code dependent.

HandleStatusCode is a VIRTUAL method so that other base class methods can directly call the HandleStatusCode virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method is PROTECTED, therefore, it can only be called by an HttpPageBaseClass method, or a method in a class derived from HttpPageBaseClass.

Implementation: The HandleStatusCode method calls the SetupHttpStatus method to set an outgoing message corresponding to the Status property.

Example:

```
HttpPageBaseClass.PreparePage PROCEDURE
CODE
IF (SELF.GotHtmlBody)
    SELF.FileLen = SELF.FileHandler.GetSize(SELF.PageFilename)
    IC:RemoveFile(SELF.HtmlFilename)
    IC:RenameFile(SELF.PageFilename, SELF.HtmlFilename)
    SELF.Http.SetServerProperty('Content-length', '' & SELF.FileLen)
ELSE
    SELF.HtmlFilename = SELF.PageFilename
END
SELF.PreparePageBody()
SELF.HandleStatusCode()           !set status message & other status dependent http
```

See Also: [HttpPageBrowserClass.HandleStatusCode](#), [SetupHttpStatus](#), [Status](#)

## Init (initialize the `HttpPageBaseClass` object)

**Init**( *httpclass*, *status*, *page*, *tempfilename* )

<b>Init</b>	Initializes the <code>HttpPageBaseClass</code> object.
<i>httpclass</i>	The label of the <code>HttpClass</code> object that provides access to the outgoing http information ( <code>ServerInfo</code> ). This is typically that same object that calls the <code>Init</code> method.
<i>status</i>	An integer constant, variable, <code>EQUATE</code> , or expression indicating the status of the Client request.
<i>page</i>	A string constant, variable, <code>EQUATE</code> , or expression containing the filename/pathname of the file to send to the Client.
<i>tempfile</i>	A string constant, variable, <code>EQUATE</code> , or expression containing the filename/pathname of the temporary file used to build the <i>page</i> file.

The **Init** method initializes the `HttpPageBaseClass` object.

Implementation:

The `Init` method sets the value of the `GotHtmlBody`, `HtmlFilename`, `HttpPageFilename`, and `Status` properties, and instantiates a new `TextOutputClass` object for the `FileHandler` property.

Example:

```
HttpClass.PreparePage PROCEDURE |
    (*HttpPageBaseClass HttpPage,SIGNED Status,<STRING Filename>)
    CODE
    SELF.ClearUp()
    SELF.HttpPage &= HttpPage
    SELF.HttpPage.Init(SELF,status,Filename,SELF.Files.GetTempFilename(Filename))
    SELF.HttpPage.PreparePage()
```

See Also:

`GotHtmlBody`, `FileHandler`, `HtmlFilename`, `HttpPageFilename`, `Status`

## Kill (shut down the HttpPageBaseClass object)

---

### Kill

The **Kill** method frees any memory allocated during the life of the object and performs any other required termination code.

Implementation: The Kill method DISPOSEs the FileHandler property.

Example:

```
HttpClass.FinishPage PROCEDURE
CODE
SELF.HttpPage.FinishPage()
IF (NOT SELF.HttpPage &= NULL)
    SELF.HttpPage.Kill()
    DISPOSE(SELF.HttpPage)
END
```

See Also: FileHandler

## PreparePage (set outgoing http items)

---

### PreparePage, VIRTUAL

The **PreparePage** method sets the final pieces of outgoing http information.

PreparePage is a VIRTUAL method so that other base class methods can directly call the PreparePage virtual method in a derived class. This lets you easily implement your own custom version of this method.

Implementation: Among other things, the PreparePage method calls the PreparePageBody and HandleStatusCode methods to add the final outgoing http information to the HttpClass.ServerInfo property.

Example:

```
HttpClass.PreparePage PROCEDURE |
    (*HttpPageBaseClass HttpPage, SIGNED Status, <STRING Filename>)
CODE
SELF.ClearUp()
SELF.HttpPage &= HttpPage
SELF.HttpPage.Init(SELF,status,Filename,SELF.Files.GetTempFilename(Filename))
SELF.HttpPage.PreparePage() !wrap up outgoing http information
```

See Also: HandleStatusCode, HttpClass.ServerInfo, PreparePageBody

## PreparePageBody (virtual to set outgoing http items)

---

### PreparePageBody, VIRTUAL, PROTECTED

The **PreparePageBody** method is a virtual placeholder to set some outgoing http information.

PreparePageBody is a VIRTUAL method so that other base class methods can directly call the PreparePageBody virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method is PROTECTED, therefore, it can only be called by an HttpPageBaseClass method, or a method in a class derived from HttpPageBaseClass.

Implementation: The HttpPageBaseClass.PreparePageBody method does nothing. It is a placeholder for derived class methods.

Example:

```
HttpPageBaseClass.PreparePage PROCEDURE
CODE
IF (SELF.GotHtmlBody)
    SELF.FileLen = SELF.FileHandler.GetSize(SELF.PageFilename)
    IC:RemoveFile(SELF.HtmlFilename)
    IC:RenameFile(SELF.PageFilename, SELF.HtmlFilename)
    SELF.Http.SetServerProperty('Content-length', '' & SELF.FileLen)
ELSE
    SELF.HtmlFilename = SELF.PageFilename
END
SELF.NowDateTime = IC:GetStrDateTime(TODAY(), CLOCK())
SELF.ExpireDateTime = SELF.NowDateTime
SELF.PreparePageBody()
SELF.HandleStatusCode()
```

See Also: [HttpPageBrowserClass.PreparePageBody](#)

## SetupHttpStatus (set outgoing http status indicators)

---

### SetupHttpStatus( *status* ), PROTECTED

**SetupHttpStatus** Sets the initial outgoing http header information.

*status* An integer constant, variable, EQUATE, or expression indicating the status of the Client request.

The **SetupHttpStatus** method sets the initial outgoing http header information.

This method is PROTECTED, therefore, it can only be called by an `HttpPageBaseClass` method, or a method in a class derived from `HttpPageBaseClass`.

Implementation:

The `SetupHttpStatus` method sets the `HttpProtocol`, `ErrorNum`, and `ErrorMsg` outgoing http items. These are the initial items interpreted by the Client browser. The `ErrorNum` and `ErrorMsg` values depend on the `Status` property.

Example:

```
HttpPageBaseClass.HandleStatusCode    PROCEDURE
CODE
SELF.SetupHttpStatus(SELF.Status)
```

See Also:

**Status**

## WritePageBody (write outgoing http body)

### WritePageBody, VIRTUAL, PROTECTED

The **WritePageBody** method writes all the outgoing http information that requires no special handling or special positioning within the http header.

WritePageBody is a VIRTUAL method so that other base class methods can directly call the WritePageBody virtual method in a derived class. This lets you easily implement your own custom version of this method.

This method is PROTECTED, therefore, it can only be called by an HttpPageBaseClass method, or a method in a class derived from HttpPageBaseClass.

#### Implementation:

The WritePageBody method loops through the HttpClass.ServerInfo property, writing out each non-special item. Special items that require specific positioning, such as HttpProtocol, ErrorNum, and ErrorMsg, are handled separately.

#### Example:

```
HttpPageBaseClass.FinishPage PROCEDURE
Protocol          CSTRING(20)
ErrorNum          CSTRING(5)
ErrorMsg          CSTRING(30)
CODE
SELF.FileHandler.CreateOpen(SELF.PageFilename)
Protocol = SELF.Http.GetServerProperty('HttpProtocol')
ErrorNum = SELF.Http.GetServerProperty('ErrorNum')
ErrorMsg = SELF.Http.GetServerProperty('ErrorMsg')
SELF.FileHandler.WriteLine(Protocol & ' ' & ErrorNum & ' ' & ErrorMsg)

SELF.WritePageBody()           !write non-special http information

SELF.FileHandler.WriteLine()
SELF.FileHandler.Close()
IF (SELF.GotHtmlBody)
    SELF.AppendFileTo(SELF.HtmlFilename, SELF.PageFilename, SELF.FileLen, TRUE)
ELSE
    SELF.AppendDefaultBody()
END
```

#### See Also:

[HttpClass.ServerInfo](#)

**INDEX**

- A**
- ActionOnAccept
    - WebControlClass Properties ..... 221
  - Active
    - WebServerClass Properties ..... 66
  - AddControl
    - WebWindowClass Methods ..... 167
  - AddControlsToLayout
    - WebControlListClass Methods ..... 135
    - WebWindowClass Methods ..... 169
  - AddEvent
    - JSEventsClass Methods ..... 481
  - AddQueueEntry
    - JSLManagerClass Methods ..... 456
  - aliases ..... 489
  - Alignment
    - WebCaptionClass Properties ..... 422
  - AllowJava
    - WebWindowBaseClass Properties ..... 140
  - AltText
    - WebHtmlImageClass Properties ..... 346
  - AppendDefaultBody
    - HttpPageBaseClass Methods ..... 613
  - AppletCount
    - HtmlClass Properties ..... 529
  - Application Broker ..... 32
  - ArgIndex
    - WebServerClass Properties ..... 66
  - Arguments
    - HttpClass Properties ..... 583
  - Authorize
    - WebWindowClass Properties ..... 162
  - AuthorizeArea
    - WebWindowClass Properties ..... 163
  - AutoSpotLink
    - WebJavaListClass Properties ..... 359
    - WebJavaStringClass Properties ..... 380
- B**
- Background
    - WebAreaClass Properties ..... 411
    - WebWindowBaseClass Properties ..... 140
  - BackImage
    - WebAreaClass Properties ..... 411
    - WebWindowBaseClass Properties ..... 140
  - BeforeResetControl
    - WebButtonClass Methods ..... 328
    - WebControlClass Methods ..... 227
    - WebHtmlCheckClass Methods ..... 271
  - BeginUpdate
    - JSLManagerClass Methods ..... 457
  - BodyFooter
    - WebWindowClass Methods ..... 170
  - BodyHeader
    - WebWindowClass Methods ..... 171
  - BorderWidth
    - WebHtmlGroupClass Properties ..... 279
    - WebHtmlOptionClass Properties ..... 291
    - WebHtmlSheetClass Properties ..... 304
    - WebWindowBaseClass Properties ..... 141
  - Broker ..... 32
    - WebClientManagerClass Properties ..... 94
    - WebServerClass Properties ..... 66
  - BrokerClass ..... 41
    - BrokerClass Methods ..... 46
    - CloseChannel ..... 47
    - GetAuthorizedInfo ..... 48
    - GetClient ..... 49
    - GetRequestArguments ..... 50
    - Init ..... 51
    - Kill ..... 52
    - OpenChannel ..... 53
    - ProcessHttpHeader ..... 54
    - SetClient ..... 55
    - SetClientBrowser ..... 55
    - TakeFile ..... 56
    - TakeHtmlPage ..... 57
    - TakeJsldata ..... 58
    - TakeUnauthorized ..... 59
  - BrokerClass Properties ..... 44
    - Client ..... 44
    - Files ..... 45
    - Http ..... 44
    - ServerName ..... 45
  - BrokerClass synopsis ..... 35
  - Browser
    - HtmlClass Properties ..... 529
    - WebClientManagerClass Properties ..... 94
  - BrowserInfo
    - HttpClass Properties ..... 584
  - BrowserManagerClass ..... 109
  - BrowserManagerClass Methods ..... 112
    - Init ..... 112
  - BrowserManagerClass Properties ..... 110
    - Kind ..... 110
    - SetNoCache ..... 110
    - SubmitFromJava ..... 111
    - SupportsStyleSheets ..... 111

<b>C</b>	
caching options .....	451
Class Libraries .....	31
Client .....	32
BrokerClass Properties .....	44
HtmlClass Properties .....	529
JSLManagerClass Properties .....	453
WebServerClass Properties .....	67
Close	
ShutDownClass Methods .....	90
TextOutputClass Methods .....	573
CloseChannel	
BrokerClass Methods .....	47
JSLManagerClass Methods .....	458
CloseImage	
WebWindowBaseClass Properties .....	141
CommandLine	
WebServerClass Properties .....	68
Connect	
WebServerClass Methods .....	74
Container	
WebControlClass Properties .....	222
Container Applet .....	555
Conventions	
documentation .....	37
Cookies	
HttpClass Properties .....	585
CopyControlsToWindow	
WebFrameClass Methods .....	120
CopyControlToWindow	
WebFrameClass Methods .....	122
CreateCaption	
WebWindowBaseClass Properties .....	141
CreateCellContents	
WebControlClass Methods .....	227
WebHotlinkClass Methods .....	398
WebHtmlButtonClass Methods .....	332
WebHtmlCheckClass Methods .....	272
WebHtmlEntryClass Methods .....	276
WebHtmlImageClass Methods .....	347
WebHtmlItemClass Methods .....	284
WebHtmlListClass Methods .....	356
WebHtmlMenuClass Methods .....	287
WebHtmlPromptClass Methods .....	298
WebHtmlRadioClass Methods .....	300
WebHtmlStringClass Methods .....	378
WebHtmlTextClass Methods .....	325
CreateCellFooter	
WebControlClass Methods .....	228
CreateCellHeader	
WebControlClass Methods .....	228
CreateChildHtml	
WebWindowBaseClass Methods .....	153
WebWindowClass Methods .....	172
CreateClose	
WebWindowBaseClass Properties .....	142
CreateColorParameters	
WebControlClass Methods .....	229
CreateDummyHtmlPage	
WebWindowClass Methods .....	173
CreateForeColorParameter	
WebControlClass Methods .....	230
CreateHtml	
LayoutHtmlClass Methods .....	513
WebCaptionClass Methods .....	423
WebClientAreaClass Methods .....	437
WebCloseButtonClass Methods .....	388
WebControlClass Methods .....	231
WebControlListClass Methods .....	136
WebHtmlGroupClass Methods .....	280
WebHtmlMenuClass Methods .....	288
WebHtmlOptionClass Methods .....	292
WebHtmlSheetClass Methods .....	305
WebHtmlTabClass Methods .....	315
WebJavaButtonClass Methods .....	337
WebJavalmageClass Methods .....	350
WebJavaListClass Methods .....	362
WebJavaStringClass Methods .....	381
WebLiteralClass Methods .....	400
WebMenuBarClass Methods .....	429
WebNullControlClass Methods .....	402
WebToolBarClass Methods .....	433
CreateHtmlExtra	
WebControlClass Methods .....	232
WebHtmlRegionClass Methods .....	302
CreateHtmlPage	
WebWindowClass Methods .....	174
CreateJsIData	
WebCloseButtonClass Methods .....	390
WebControlClass Methods .....	232
WebHtmlCheckClass Methods .....	273
WebHtmlEntryClass Methods .....	277
WebHtmlListClass Methods .....	357
WebHtmlOptionClass Methods .....	293
WebHtmlTabClass Methods .....	316
WebHtmlTextClass Methods .....	327
WebJavaButtonClass Methods .....	339
WebJavalmageClass Methods .....	352
WebJavaListClass Methods .....	364
WebJavaStringClass Methods .....	382
WebNullControlClass Methods .....	402
WebWindowClass Methods .....	175
CreateOpen	

- HtmlClass Methods ..... 535
  - TextOutputClass Methods ..... 574
  - CreatePageFooter
    - WebWindowClass Methods ..... 176
  - CreatePageHeader
    - WebWindowClass Methods ..... 176
  - CreateParams
    - WebCloseButtonClass Methods ..... 391
    - WebControlClass Methods ..... 233
    - WebHtmlTabClass Methods ..... 317
    - WebJavaButtonClass Methods ..... 339
    - WebJavalmageClass Methods ..... 353
    - WebJavaListClass Methods ..... 365
    - WebJavaStringClass Methods ..... 383
  - CreateTabControl
    - WebHtmlSheetClass Methods ..... 307
  - CreateToolBar
    - WebWindowBaseClass Properties ..... 142
  - CreateUnauthorizedPage
    - WebWindowClass Methods ..... 177
  - CurSubmit
    - WebServerClass Properties ..... 67
- D**
- DefaultButton
    - WebWindowBaseClass Properties ..... 143
  - DefaultButtonNeeded
    - WebWindowBaseClass Properties ..... 143
  - DialogPageBackColor
    - WebServerClass Properties ..... 68
  - DialogPageImage
    - WebServerClass Properties ..... 68
  - DialogWinBackColor
    - WebServerClass Properties ..... 68
  - DialogWinImage
    - WebServerClass Properties ..... 69
  - DisabledAction
    - WebControlClass Properties ..... 222
    - WebWindowBaseClass Properties ..... 144
  - Documentation Conventions ..... 37
  - DoSetChildDefaults
    - WebControlClass Methods ..... 233
  - dynamic HTML ..... 451
- E**
- Event
    - SubmitItem Properties ..... 522
  - EventActionQ
    - WebJavaListClass Properties ..... 359
  - EventQ
- JsEventsClass Properties ..... 479
  - ExpireDateTime
    - HttpPageBaseClass Properties ..... 607
  - Extra
    - SubmitItem Properties ..... 522
- F**
- Feq
    - SubmitItem Properties ..... 522
    - WebControlClass Properties ..... 223
  - Feq2Id
    - WebClientManagerClass Methods ..... 97
  - FileHandler
    - HttpPageBaseClass Properties ..... 607
  - FileLen
    - HttpPageBaseClass Properties ..... 608
  - Filename
    - WebJavalmageClass Properties ..... 349
  - Files
    - BrokerClass Properties ..... 45
    - HtmlClass Properties ..... 530
    - HttpClass Properties ..... 586
    - JSLManagerClass Properties ..... 453
    - WebServerClass Properties ..... 69
    - WebWindowBaseClass Properties ..... 144
  - FinishPage
    - HttpClass Methods ..... 591
    - HttpPageBaseClass Methods ..... 614
  - FirstControl
    - HtmlClass Properties ..... 530
  - FirstSelectable
    - HtmlClass Properties ..... 530
  - Format
    - WebJavaListClass Properties ..... 360
  - FormatBorderWidth
    - WebWindowBaseClass Properties ..... 145
  - FrameWindow
    - WebFrameClass Properties ..... 119
  - FromQ
    - WebJavaListClass Properties ..... 360
- G**
- GetAlias
    - WebFilesClass Methods ..... 492
  - GetAlignText
    - WebControlClass Methods ..... 234
  - GetAppletType
    - WebCloseButtonClass Methods ..... 392
    - WebControlClass Methods ..... 234
    - WebHtmlTabClass Methods ..... 318

WebJavaButtonClass Methods .....	340	GetControlReference	
WebJavaImageClass Methods .....	354	HtmlClass Methods .....	540
WebJavaListClass Methods .....	366	GetCookie	
WebJavaStringClass Methods .....	384	HttpClass Methods .....	594
WebNullControlClass Methods .....	403	GetCreateClose	
WebToolBarClass Methods .....	434	WebWindowBaseClass Methods .....	155
GetArguments		WebWindowClass Methods .....	188
HttpClass Methods .....	591	GetDirectory	
GetAuthorized		WebFilesClass Methods .....	493
WebWindowClass Methods .....	179	GetEventAction	
GetAuthorizedInfo		WebControlClass Methods .....	239
BrokerClass Methods .....	48	WebJavaListClass Methods .....	367
HttpClass Methods .....	592	WebJavaToggleButtonClass Methods .....	343
GetBackgroundColor		GetEventNumber	
WebAreaClass Methods .....	412	JSLEventsClass Methods .....	484
WebClientAreaClass Methods .....	438	GetEventString	
WebControlClass Methods .....	235	JSLEventsClass Methods .....	485
WebListClass Methods .....	355	GetFilename	
WebWindowBaseClass Methods .....	154	WebFilesClass Methods .....	494
WebWindowClass Methods .....	180	WebJavaButtonClass Methods .....	342
GetBackgroundImage		GetFirstChild	
WebWindowBaseClass Methods .....	154	WebWindowBaseClass Methods .....	157
WebWindowClass Methods .....	181	WebWindowClass Methods .....	187
GetBrowserProperty		GetFont	
HttpClass Methods .....	593	WebAreaClass Methods .....	414
GetButtonInClientArea		WebControlClass Methods .....	240
WebWindowClass Methods .....	182	GetFontChanged	
GetCanDisable		HtmlClass Methods .....	536
WebControlClass Methods .....	236	GetFontStyle	
WebJavaButtonClass Methods .....	341	HtmlClass Methods .....	537
GetCellAttributes		GetHasHotkey	
WebAreaClass Methods .....	413	WebButtonClass Methods .....	330
WebCaptionClass Methods .....	424	WebControlClass Methods .....	240
WebControlClass Methods .....	237	WebHtmlCheckClass Methods .....	274
WebHtmlMenuClass Methods .....	289	WebHtmlGroupClass Methods .....	281
WebHtmlStringClass Methods .....	379	WebHtmlPromptClass Methods .....	299
WebLiteralClass Methods .....	401	WebHtmlRadioClass Methods .....	301
WebNullControlClass Methods .....	404	WebHtmlTabClass Methods .....	319
GetChildren		GetHelpHandler	
WebWindowBaseClass Methods .....	156	WebWindowBaseClass Methods .....	157
WebWindowClass Methods .....	183	WebWindowClass Methods .....	189
GetChoiceChanged		GetHelpReference	
WebControlClass Methods .....	238	WebWindowBaseClass Methods .....	158
GetClient		WebWindowClass Methods .....	190
BrokerClass Methods .....	49	GetHelpTarget	
GetCloneFreq		WebWindowBaseClass Methods .....	158
WebCloseButtonClass Methods .....	393	WebWindowClass Methods .....	192
GetControl		GetInternetEnabled	
WebWindowBaseClass Methods .....	155	WebServerClass Methods .....	75
WebWindowClass Methods .....	184	GetIsChild	
GetControlInfo		WebControlClass Methods .....	241
WebWindowClass Methods .....	186	WebHtmlSheetClass Methods .....	308

WebHtmlTabClass Methods .....	320	WebWindowClass Methods .....	194
WebNullControlClass Methods .....	405	GetSize	
GetLevel		TextOutputClass Methods .....	575
WebControlClass Methods .....	242	GetTableAttributes	
GetMenuBarFeq		WebControlClass Methods .....	245
WebFrameClass Methods .....	123	WebHtmlOptionClass Methods .....	294
WebWindowBaseClass Methods .....	158	WebHtmlSheetClass Methods .....	309
WebWindowClass Methods .....	191	WebWindowClass Methods .....	195
GetNameAttribute		GetTargetSecurity	
WebControlClass Methods .....	242	WebWindowClass Methods .....	195
GetPageImage		GetTempFilename	
WebWindowBaseClass Methods .....	159	WebFilesClass Methods .....	500
WebWindowClass Methods .....	192	GetText	
GetParentBackgroundColor		WebCaptionClass Methods .....	426
WebControlClass Methods .....	243	WebControlClass Methods .....	245
GetPixelsX		GetToolBarFeq	
HtmlClass Methods .....	538	WebFrameClass Methods .....	124
GetPixelsY		WebWindowBaseClass Methods .....	160
HtmlClass Methods .....	539	WebWindowClass Methods .....	196
GetPosition		GetToolBarMode	
WebCaptionClass Methods .....	425	WebWindowBaseClass Methods .....	160
WebClientAreaClass Methods .....	439	WebWindowClass Methods .....	199
WebCloseButtonClass Methods .....	394	GetUseChanged	
WebControlClass Methods .....	244	WebControlClass Methods .....	246
WebHtmlTabClass Methods .....	322	GetVisible	
WebMenuBarClass Methods .....	430	WebAreaClass Methods .....	415
WebToolBarClass Methods .....	435	WebButtonClass Methods .....	331
GetProgramRef		WebCaptionClass Methods .....	427
WebFilesClass Methods .....	495	WebCloseButtonClass Methods .....	395
GetPublicDirectory		WebControlClass Methods .....	247
WebFilesClass Methods .....	496	WebHtmlItemClass Methods .....	285
GetQuotedText		WebHtmlMenuClass Methods .....	290
WebControlClass Methods .....	245	WebHtmlTabClass Methods .....	323
GetReadyForPage		WebMenuBarClass Methods .....	431
WebServerClass Methods .....	76	WebNullControlClass Methods .....	406
GetRelativeFilename		WebToolBarClass Methods .....	436
WebFilesClass Methods .....	498	GetWebActiveFrame	
GetRequestArguments		WebWindowBaseClass Methods .....	161
BrokerClass Methods .....	50	WebWindowClass Methods .....	198
GetRequestedWholePage		GotCommandLine	
WebServerClass Methods .....	77	WebServerClass Properties .....	69
GetSendWholePage		GotHtmlBody	
WebServerClass Methods .....	78	HttpPageBaseClass Properties .....	608
GetSeparateSecure		GroupBorderWidth	
WebFilesClass Methods .....	499	WebWindowBaseClass Properties .....	145
GetServerProperty			
HttpClass Methods .....	595	<b>H</b>	
GetShowMenuBar		Halt	
WebWindowBaseClass Methods .....	159	WebServerClass Methods .....	79
WebWindowClass Methods .....	194	HandleStatusCode	
GetShowToolBar		HttpPageBaseClass Methods .....	615
WebWindowBaseClass Methods .....	159		

Header Files .....	36	WriteTableNewCol .....	567
Height		WriteTableNewRow .....	567
WebCloseButtonClass Properties .....	387	WriteText .....	568
HelpDocument		HtmlClass Properties .....	529
WebWindowBaseClass Properties .....	145	AppletCount .....	529
HelpEnabled		Browser .....	529
WebWindowBaseClass Properties .....	146	Client .....	529
HelpRelative		Files .....	530
WebWindowBaseClass Properties .....	146	FirstControl .....	530
HelpStyle		FirstSelectable .....	530
WebWindowBaseClass Properties .....	146	JavaLibraryCab .....	531
Html		JavaLibraryZip .....	531
WebReportClass Properties .....	445	Option .....	532
HTML code generator .....	30, 31	UseFonts .....	532
HtmlClass .....	527	HtmlFilename	
HtmlClass Methods .....	533	HttpPageBaseClass Properties .....	609
CreateOpen .....	535	HtmlOption	
GetControlReference .....	540	WebWindowBaseClass Properties .....	147
GetFontChanged .....	536	HtmlPreview .....	443
GetFontStyle .....	537	HtmlTarget	
GetPixelsX .....	538	WebWindowClass Properties .....	163
GetPixelsY .....	539	Http	
Init .....	541	BrokerClass Properties .....	44
Kill .....	541	HttpPageBaseClass Properties .....	609
PopFont .....	542	HttpClass .....	581
PushFont .....	543	HttpClass Methods .....	589
TakeNewControl .....	544	FinishPage .....	591
WriteAppletDimParameter .....	545	GetArguments .....	591
WriteAppletFilenameParameter .....	546	GetAuthorizedInfo .....	592
WriteAppletFontParameter .....	547	GetBrowserProperty .....	593
WriteAppletFooter .....	548	GetCookie .....	594
WriteAppletHeader .....	549	GetServerProperty .....	595
WriteAppletOptParameter .....	550	Init .....	596
WriteAppletParameter .....	551	Kill .....	596
WriteAppletUAID .....	552	PreparePage .....	597
WriteChildAppletFooter .....	553	PreparePageForBrowser .....	598
WriteChildAppletHeader .....	554	PreparePageForJava .....	599
WriteContainerAppletHeader .....	555	PrepareUnauthorized .....	600
WriteControlFooter .....	556	ProcessHeader .....	601
WriteControlHeader .....	556	SendCookies .....	601
WriteEventHandler .....	557	SetBrowserProperty .....	602
WriteFontFooter .....	559	SetCookie .....	603
WriteFontHeader .....	559	SetProcName .....	605
WriteFormFooter .....	560	SetProgName .....	604
WriteFormHeader .....	560	SetServerProperty .....	606
WriteJavaScript .....	561	HttpClass Properties .....	583
WriteOnFocusHandler .....	562	Arguments .....	583
WriteRefreshTimer .....	563	BrowserInfo .....	584
WriteSpace .....	564	Cookies .....	585
WriteSubmitApplet .....	565	Files .....	586
WriteTableFooter .....	566	HttpPage .....	586
WriteTableHeader .....	566	ProcName .....	587

- ProgName ..... 587
  - ServerInfo ..... 588
  - HttpPage
    - HttpClass Properties ..... 586
  - HttpPageBaseClass ..... 581
  - HttpPageBaseClass Methods ..... 612
    - AppendDefaultBody ..... 613
    - FinishPage ..... 614
    - HandleStatusCode ..... 615
    - Init ..... 616
    - Kill ..... 617
    - PreparePage ..... 617
    - PreparePageBody ..... 618
    - SetupHttpStatus ..... 619
    - WritePageBody ..... 620
  - HttpPageBaseClass Properties ..... 607
    - ExpireDateTime ..... 607
    - FileHandler ..... 607
    - FileLen ..... 608
    - GotHtmlBody ..... 608
    - HtmlFilename ..... 609
    - Http ..... 609
    - NowDateTime ..... 610
    - PageFilename ..... 610
    - Status ..... 611
  - HttpProtocol ..... 588
  - Hyper-Text Transfer Protocol ..... 581
- I**
- IBC Library ..... 30, 31
  - IBC Library Reference ..... 30
  - IC:CurControl ..... 220
  - ICBROKER.INC ..... 42
  - ICCLIENT.INC ..... 93, 109
  - ICCNTRLS.INC ..... 270, 410
  - ICEVENTS.INC ..... 478
  - ICFILES.INC ..... 490
  - ICHTML.INC ..... 528
  - ICHTTP.INC ..... 582
  - ICJSL.INC ..... 452
  - ICLAYOUT.INC ..... 220, 510
  - ICREPORT.INC ..... 444
  - ICREPORT.TRN ..... 444
  - ICSERVER.INC ..... 64, 521
  - ICSERVER.TRN ..... 64
  - ICTXTOUT.INC ..... 528, 571
  - ICWINDOW.INC ..... 117, 133, 220
  - ICWINDOW.TRN ..... 133
  - Init
    - BrokerClass Methods ..... 51
    - BrowserManagerClass Methods ..... 112
    - HtmlClass Methods ..... 541
    - HttpClass Methods ..... 596
    - HttpPageBaseClass Methods ..... 616
    - JSLEventsClass Methods ..... 482
    - JSLManagerClass Methods ..... 459
    - LayoutHtmlClass Methods ..... 514
    - WebAreaClass Methods ..... 416
    - WebButtonClass Methods ..... 333
    - WebCaptionClass Methods ..... 428
    - WebClientManagerClass Methods ..... 98
    - WebCloseButtonClass Methods ..... 396
    - WebControlClass Methods ..... 248
    - WebControlListClass Methods ..... 138
    - WebFilesClass Methods ..... 501
    - WebHtmlGroupClass Methods ..... 282
    - WebHtmlOptionClass Methods ..... 295
    - WebHtmlSheetClass Methods ..... 310
    - WebJavaListClass Methods ..... 368
    - WebJavaStringClass Methods ..... 385
    - WebMenuBarClass Methods ..... 432
    - WebReportClass Methods ..... 447
    - WebServerClass Methods ..... 80
    - WebWindowClass Methods ..... 200
  - Insert
    - LayoutHtmlClass Methods ..... 515
  - Internet Builder Class (IBC) Library ..... 30
  - Internet Builder Class Synopsis ..... 35
  - Internet Builder Classes ..... 31
  - Internet Connect Terms and Concepts ..... 32
  - InternetDataSinkClass ..... 41
  - IP
    - WebClientManagerClass Properties ..... 95
  - IsCentered
    - WebWindowClass Properties ..... 163
  - IsDynamic
    - WebControlClass Properties ..... 223
  - IsEnabled
    - WebHtmlTabClass Properties ..... 314
    - WebJavaButtonClass Properties ..... 336
  - IsSecure
    - WebWindowClass Properties ..... 164
  - IsSplash
    - WebWindowBaseClass Properties ..... 147
- J**
- Java events ..... 477
  - Java Support Library ..... 32
  - Java Support Library (JSL) ..... 451
  - Java-enabled browser ..... 32
  - JavaLibraryCab
    - HtmlClass Properties ..... 531

JavaLibraryPath	
WebServerClass Properties .....	70
JavaLibraryZip	
HtmlClass Properties .....	531
JSL .....	451
Jsl	
WebClientManagerClass Properties .....	95
JslEventsClass .....	477
JSLEventsClass Methods .....	480
AddEvent .....	481
GetEventNumber .....	484
GetEventString .....	485
Init .....	482
Kill .....	483
JslEventsClass Properties .....	479
EventQ .....	479
JSLManagerClass .....	451
JSLManagerClass Methods .....	455
AddQueueEntry .....	456
BeginUpdate .....	457
CloseChannel .....	458
Init .....	459
Kill .....	459
OpenChannel .....	460
RemoveAllQueueEntries .....	462
RemoveQueueEntries .....	463
ScrollQueueDown .....	464
ScrollQueueUp .....	465
SelectControl .....	466
SetAttribute .....	467
SetAttributeFilename .....	468
SetAttributeLong .....	469
SetChecked .....	470
SetIconAttribute .....	471
SetListChoice .....	472
SetQueueEntry .....	473
SetValue .....	474
JSLManagerClass Properties .....	453
Client .....	453
Files .....	453
Security .....	454
Target .....	454

**K**

Kill	
BrokerClass Methods .....	52
HtmlClass Methods .....	541
HttpClass Methods .....	596
HttpPageBaseClass Methods .....	617
JSLEventsClass Methods .....	483
JSLManagerClass Methods .....	459

LayoutHtmlClass Methods .....	516
WebAreaClass Methods .....	417
WebClientManagerClass Methods .....	99
WebControlClass Methods .....	249
WebControlListClass Methods .....	138
WebFilesClass Methods .....	502
WebJavaListClass Methods .....	369
WebReportClass Methods .....	447
WebServerClass Methods .....	81
WebWindowClass Methods .....	200, 201
Kind	
BrowserManagerClass Properties .....	110

**L**

LastText	
WebJavaStringClass Properties .....	380
LayoutHtmlClass .....	509
LayoutHtmlClass Methods .....	512
CreateHtml .....	513
Init .....	514
Insert .....	515
Kill .....	516
SetCell .....	517
LayoutHtmlClass Properties .....	511
SnapX .....	511
SnapY .....	511
Style .....	511
LoadImage	
WebFilesClass Methods .....	503
LocalFont	
WebAreaClass Properties .....	411

**M**

menu merging .....	117
MenuBarFeq	
WebFrameClass Properties .....	119
WebWindowBaseClass Properties .....	148
MenuBarType	
WebWindowBaseClass Properties .....	148
merge menus and toolbars .....	117

**N**

Name	
SubmitItem Properties .....	523
NewValue	
SubmitItem Properties .....	523
NextHtmlPage	
WebClientManagerClass Methods .....	100
NowDateTime	

- HttpPageBaseClass Properties ..... 610
- NumPages
- WebReportClass Properties ..... 446
- O**
- Object Oriented system ..... 31
- Open
- TextOutputClass Methods ..... 576
- OpenChannel
- BrokerClass Methods ..... 53
- JSLManagerClass Methods ..... 460
- Option
- HtmlClass Properties ..... 532
- OptionBorderWidth
- WebWindowBaseClass Properties ..... 148
- OwnerWindow
- WebControlClass Properties ..... 223
- P**
- PageBackground
- WebWindowBaseClass Properties ..... 149
- PageFilename
- HttpPageBaseClass Properties ..... 610
- PageImage
- WebWindowBaseClass Properties ..... 149
- PageToReturnTo
- WebServerClass Properties ..... 70
- ParentFeq
- WebControlClass Properties ..... 224
- Partial Page Updates ..... 451
- persistent communication ..... 33
- PopFont
- HtmlClass Methods ..... 542
- WebControlClass Methods ..... 250
- PreparePage
- HttpClass Methods ..... 597
- HttpPageBaseClass Methods ..... 617
- PreparePageBody
- HttpPageBaseClass Methods ..... 618
- PreparePageForBrowser
- HttpClass Methods ..... 598
- PreparePageForJava
- HttpClass Methods ..... 599
- PrepareUnauthorized
- HttpClass Methods ..... 600
- Preview
- WebReportClass Methods ..... 448
- ProcessHeader
- HttpClass Methods ..... 601
- ProcessHttpHeader
- BrokerClass Methods ..... 54
- ProcName
- HttpClass Properties ..... 587
- ProgName
- HttpClass Properties ..... 587
- ProgramName
- WebServerClass Properties ..... 71
- PushFont
- HtmlClass Methods ..... 543
- WebAreaClass Methods ..... 418
- WebControlClass Methods ..... 251
- Q**
- Q
- WebReportClass Properties ..... 445
- QueueActionQ
- WebJavaListClass Properties ..... 361
- Quit
- WebServerClass Methods ..... 82
- R**
- RealParentFeq
- WebControlClass Properties ..... 224
- RefreshDisabled
- WebControlClass Methods ..... 252
- RemoveAll
- WebFilesClass Methods ..... 504
- RemoveAllQueueEntries
- JSLManagerClass Methods ..... 462
- RemoveQueueEntries
- JSLManagerClass Methods ..... 463
- Reports ..... 443
- Reset
- SubmitItem Methods ..... 524
- ResetControl
- WebButtonClass Methods ..... 331
- WebCloseButtonClass Methods ..... 397
- WebControlClass Methods ..... 253
- WebHtmlButtonClass Methods ..... 334
- WebHtmlCheckClass Methods ..... 275
- WebHtmlEntryClass Methods ..... 278
- WebHtmlItemClass Methods ..... 286
- WebHtmlListClass Methods ..... 358
- WebHtmlOptionClass Methods ..... 296
- WebHtmlSheetClass Methods ..... 311
- WebHtmlTabClass Methods ..... 323
- WebHtmlTextClass Methods ..... 327
- WebJavaButtonClass Methods ..... 342
- WebJavaListClass Methods ..... 370
- ResetFromControls
- WebWindowClass Methods ..... 200

ResetFromQueue		SetCentered	
WebControlClass Methods .....	254	WebWindowClass Methods .....	202
WebJavaListClass Methods .....	371	SetChecked	
<b>S</b>		JSLManagerClass Methods .....	470
ScrollQueueDown		SetChildDefaults	
JSLManagerClass Methods .....	464	WebControlClass Methods .....	256
ScrollQueueUp		WebHtmlImageClass Methods .....	348
JSLManagerClass Methods .....	465	WebHtmlSheetClass Methods .....	313
Security		WebWindowClass Methods .....	202
JSLManagerClass Properties .....	454	SetClient	
SelectControl		BrokerClass Methods .....	55
JSLManagerClass Methods .....	466	SetClientBrowser	
SelectTarget		BrokerClass Methods .....	55
WebFilesClass Methods .....	505	SetCookie	
SendCookies		HttpClass Methods .....	603
HttpClass Methods .....	601	SetDescription	
SentHtml		WebControlClass Methods .....	257
WebWindowClass Properties .....	164	WebHtmlImageClass Methods .....	348
Server		SetDialogPageBackground	
WebReportClass Properties .....	446	WebServerClass Methods .....	83
WebWindowBaseClass Properties .....	149	SetDialogWinBackground	
ServerInfo		WebServerClass Methods .....	84
HttpClass Properties .....	588	SetDirty	
ServerName		WebJavaListClass Methods .....	372
BrokerClass Properties .....	45	SetEventAction	
SetAttribute		WebControlClass Methods .....	258
JSLManagerClass Methods .....	467	WebJavaListClass Methods .....	373
SetAttributeFilename		SetFont	
JSLManagerClass Methods .....	468	WebAreaClass Methods .....	420
SetAttributeLong		SetFormatOptions	
JSLManagerClass Methods .....	469	WebWindowClass Methods .....	203
SetAutoSpotLink		SetHelpDocument	
WebControlClass Methods .....	255	WebWindowClass Methods .....	204
WebJavaListClass Methods .....	372	SetHelpURL	
WebJavaStringClass Methods .....	386	WebWindowClass Methods .....	205
SetBackground		SetIconAttribute	
WebAreaClass Methods .....	419	JSLManagerClass Methods .....	471
WebWindowClass Methods .....	201	SetListChoice	
SetBorderWidth		JSLManagerClass Methods .....	472
WebControlClass Methods .....	255	SetNewPageDisable	
WebHtmlGroupClass Methods .....	283	WebServerClass Methods .....	86
WebHtmlOptionClass Methods .....	297	SetNextAction	
WebHtmlSheetClass Methods .....	312	WebServerClass Methods .....	87
SetBreakable		SetNoCache	
WebControlClass Methods .....	256	BrowserManagerClass Properties .....	110
WebStringClass Methods .....	377	SetNumPages	
SetBrowserProperty		WebReportClass Methods .....	448
HttpClass Methods .....	602	SetPageBackground	
SetCell		WebWindowClass Methods .....	206
LayoutHtmlClass Methods .....	517	SetParentDefaults	
		WebAreaClass Methods .....	421
		WebControlClass Methods .....	260

- WebControlListClass Methods ..... 139
  - WebHtmlRegionClass Methods ..... 303
  - WebHtmlTabClass Methods ..... 324
  - SetPassword
    - WebWindowClass Methods ..... 207
  - SetProcName
    - HttpClass Methods ..... 605
  - SetProgName
    - HttpClass Methods ..... 604
  - SetQueue
    - WebControlClass Methods ..... 261
    - WebJavaListClass Methods ..... 375
  - SetQueueEntry
    - JSLManagerClass Methods ..... 473
  - SetSendWholePage
    - WebServerClass Methods ..... 85
  - SetServerProperty
    - HttpClass Methods ..... 606
  - SetSplash
    - WebWindowClass Methods ..... 208
  - SetTimer
    - WebWindowClass Methods ..... 209
  - SetupHttpStatus
    - HttpPageBaseClass Methods ..... 619
  - SetValue
    - JSLManagerClass Methods ..... 474
  - SheetBorderWidth
    - WebWindowBaseClass Properties ..... 150
  - ShutDownClass Methods ..... 90
    - Close ..... 90
  - SnapX
    - LayoutHtmlClass Properties ..... 511
    - WebWindowBaseClass Properties ..... 150
  - SnapY
    - LayoutHtmlClass Properties ..... 511
    - WebWindowBaseClass Properties ..... 150
  - Started
    - WebJavaListClass Properties ..... 361
  - Status
    - HttpPageBaseClass Properties ..... 611
  - Style
    - LayoutHtmlClass Properties ..... 511
  - Submit Strings ..... 521
  - SubmitFromJava
    - BrowserManagerClass Properties ..... 111
  - SubmitItem Methods ..... 524
    - Reset ..... 524
  - SubmitItem Properties ..... 522
    - Event ..... 522
    - Extra ..... 522
    - Feq ..... 522
    - Name ..... 523
    - NewValue ..... 523
  - SubmitItemClass ..... 67, 521
  - SupportsStyleSheets
    - BrowserManagerClass Properties ..... 111
  - SuppressControl
    - WebWindowClass Methods ..... 210
  - Synopsis ..... 35
  - Syntax Diagrams ..... 37
- ## T
- TakeCreatePage
    - WebWindowClass Methods ..... 211
  - TakeEvent
    - WebFrameClass Methods ..... 125
    - WebServerClass Methods ..... 88
    - WebWindowClass Methods ..... 212
  - TakeFile
    - BrokerClass Methods ..... 56
    - WebClientManagerClass Methods ..... 100
  - TakeHtmlPage
    - BrokerClass Methods ..... 57
    - WebClientManagerClass Methods ..... 101
  - TakeJsData
    - BrokerClass Methods ..... 58
    - WebClientManagerClass Methods ..... 103
  - TakeNewControl
    - HtmlClass Methods ..... 544
  - TakePageSent
    - WebServerClass Methods ..... 89
  - TakeRequest
    - WebWindowClass Methods ..... 213
  - TakeUnauthorized
    - BrokerClass Methods ..... 59
    - WebClientManagerClass Methods ..... 105
  - TakeUnknownSubmit
    - WebWindowClass Methods ..... 214
  - Target
    - JSLManagerClass Properties ..... 454
  - temporary files ..... 489
  - Text
    - WebLiteralClass Properties ..... 399
  - TextOutPutClass ..... 571
  - TextOutputClass Methods ..... 572
    - Close ..... 573
    - CreateOpen ..... 574
    - GetSize ..... 575
    - Open ..... 576
    - Write ..... 577
    - WriteIn ..... 578
  - Timeout
    - WebServerClass Properties ..... 71

TimerAction		
WebWindowBaseClass Properties	151	
TimerDelay		
WebWindowBaseClass Properties	151	
TitleContents		
WebWindowClass Methods	215	
toolbar merging	117	
ToolbarFeq		
WebFrameClass Properties	119	
WebWindowBaseClass Properties	152	
<b>U</b>		
UpdateCopyChoice		
WebControlClass Methods	261	
UpdateCopyUse		
WebControlClass Methods	262	
UpdateState		
WebJavaListClass Methods	376	
UseFonts		
HtmlClass Properties	532	
Using the IBC Library	32	
<b>V</b>		
ValidatePassword		
WebWindowClass Methods	216	
<b>W</b>		
wallpaper	159, 181, 192	
Web-enable applications	31	
Web-enabled applications	30	
Web-enabled Clarion application	32	
WebAreaClass	409	
WebAreaClass Methods	412	
GetBackgroundColor	412	
GetCellAttributes	413	
GetFont	414	
GetVisible	415	
Init	416	
Kill	417	
PushFont	418	
SetBackground	419	
SetFont	420	
SetParentDefaults	421	
WebAreaClass Properties	411	
Background	411	
BackImage	411	
LocalFont	411	
WebButtonClass Methods	328	
BeforeResetControl	328	
GetHasHotkey	330	
GetVisible	331	
Init	333	
ResetControl	331	
WebCaptionClass Methods	423	
CreateHtml	423	
GetCellAttributes	424	
GetPosition	425	
GetText	426	
GetVisible	427	
Init	428	
WebCaptionClass Properties	422	
Alignment	422	
WebClientAreaClass Methods	437	
CreateHtml	437	
GetBackgroundColor	438	
GetPosition	439	
WebClientManagerClass	93	
WebClientManagerClass Methods	96	
Feq2Id	97	
Init	98	
Kill	99	
NextHtmlPage	100	
TakeFile	100	
TakeHtmlPage	101	
TakeJsldata	103	
TakeUnauthorized	105	
WebClientManagerClass Properties	94	
Broker	94	
Browser	94	
IP	95	
Jsl	95	
WebClientManagerClass synopsis	35	
WebCloseButtonClass Methods	388	
CreateHtml	388	
CreateJsldata	390	
CreateParams	391	
GetAppletType	392	
GetCloneFeq	393	
GetPosition	394	
GetVisible	395	
Init	396	
ResetControl	397	
WebCloseButtonClass Properties	387	
Height	387	
Width	387	
X	387	
Y	387	
WebControlClass	219	
WebControlClass Derived Classes	263	
WebControlClass Methods	225	

BeforeResetControl .....	227	IsDynamic .....	223
CreateCellContents .....	227	OwnerWindow .....	223
CreateCellFooter .....	228	ParentFeq .....	224
CreateCellHeader .....	228	RealParentFeq .....	224
CreateColorParameters .....	229	WebControlListClass Methods .....	134
CreateForeColorParameter .....	230	AddControlsToLayout .....	135
CreateHtml .....	231	CreateHtml .....	136
CreateHtmlExtra .....	232	Init .....	138
CreateJsldata .....	232	Kill .....	138
CreateParams .....	233	SetParentDefaults .....	139
DoSetChildDefaults .....	233	WebFilesClass .....	489
GetAlignText .....	234	WebFilesClass Methods .....	491
GetAppletType .....	234	GetAlias .....	492
GetBackgroundColor .....	235	GetDirectory .....	493
GetCanDisable .....	236	GetFilename .....	494
GetCellAttributes .....	237	GetProgramRef .....	495
GetChoiceChanged .....	238	GetPublicDirectory .....	496
GetEventAction .....	239	GetRelativeFilename .....	498
GetFont .....	240	GetSeparateSecure .....	499
GetHasHotkey .....	240	GetTempFilename .....	500
GetIsChild .....	241	Init .....	501
GetLevel .....	242	Kill .....	502
GetNameAttribute .....	242	LoadImage .....	503
GetParentBackgroundColor .....	243	RemoveAll .....	504
GetPosition .....	244	SelectTarget .....	505
GetQuotedText .....	245	WebFrameClass .....	117
GetTableAttributes .....	245	WebFrameClass Methods .....	120
GetText .....	245	CopyControlsToWindow .....	120
GetUseChanged .....	246	CopyControlToWindow .....	122
GetVisible .....	247	GetMenubarFeq .....	123
Init .....	248	GetToolbarFeq .....	124
Kill .....	249	TakeEvent .....	125
PopFont .....	250	WebFrameClass Properties .....	119
PushFont .....	251	FrameWindow .....	119
RefreshDisabled .....	252	MenubarFeq .....	119
ResetControl .....	253	ToolbarFeq .....	119
ResetFromQueue .....	254	WebHotlinkClass Methods .....	398
SetAutoSpotLink .....	255	CreateCellContents .....	398
SetBorderWidth .....	255	WebHtmlButtonClass Methods .....	332
SetBreakable .....	256	CreateCellContents .....	332
SetChildDefaults .....	256	ResetControl .....	334
SetDescription .....	257	WebHtmlCheckClass Methods .....	271
SetEventAction .....	258	BeforeResetControl .....	271
SetParentDefaults .....	260	CreateCellContents .....	272
SetQueue .....	261	CreateJsldata .....	273
UpdateCopyChoice .....	261	GetHasHotkey .....	274
UpdateCopyUse .....	262	ResetControl .....	275
WebControlClass Properties .....	221	WebHtmlEntryClass Methods .....	276
ActionOnAccept .....	221	CreateCellContents .....	276
Container .....	222	CreateJsldata .....	277
DisabledAction .....	222	ResetControl .....	278
Feq .....	223	WebHtmlGroupClass Methods .....	280

CreateHtml .....	280	WebHtmlSheetClass Properties .....	304
GetHasHotkey .....	281	BorderWidth .....	304
Init .....	282	WebHtmlStringClass Methods .....	378
SetBorderWidth .....	283	CreateCellContents .....	378
WebHtmlGroupClass Properties .....	279	GetCellAttributes .....	379
BorderWidth .....	279	WebHtmlTabClass Methods .....	315
WebHtmlImageClass Methods .....	347	CreateHtml .....	315
CreateCellContents .....	347	CreateJsldata .....	316
SetChildDefaults .....	348	CreateParams .....	317
SetDescription .....	348	GetAppletType .....	318
WebHtmlImageClass Properties .....	346	GetHasHotkey .....	321
AltText .....	346	GetIsChild .....	320
WebHtmlItemClass Methods .....	284	GetPosition .....	322
CreateCellContents .....	284	GetVisible .....	323
GetVisible .....	285	ResetControl .....	323
ResetControl .....	286	SetParentDefaults .....	324
WebHtmlListClass Methods .....	356	WebHtmlTabClass Properties .....	314
CreateCellContents .....	356	IsEnabled .....	314
CreateJsldata .....	357	WebHtmlTextClass Methods .....	325
ResetControl .....	358	CreateCellContents .....	325
WebHtmlMenuClass Methods .....	287	CreateJsldata .....	327
CreateCellContents .....	287	ResetControl .....	327
CreateHtml .....	288	WebImageClass .....	345
GetCellAttributes .....	289	WebJavaButtonClass Methods .....	336
GetVisible .....	290	CreateHtml .....	337
WebHtmlOptionClass Methods .....	292	CreateJsldata .....	339
CreateHtml .....	292	CreateParams .....	339
CreateJsldata .....	293	GetAppletType .....	340
GetTableAttributes .....	294	GetCanDisable .....	341
Init .....	295	GetFilename .....	342
ResetControl .....	296	ResetControl .....	342
SetBorderWidth .....	297	WebJavaButtonClass Properties .....	336
WebHtmlOptionClass Properties .....	291	IsEnabled .....	336
BorderWidth .....	291	WebJavaImageClass Methods .....	350
WebHtmlPromptClass Methods .....	298	CreateHtml .....	350
CreateCellContents .....	298	CreateJsldata .....	352
GetHasHotkey .....	299	CreateParams .....	353
WebHtmlRadioClass Methods .....	300	GetAppletType .....	354
CreateCellContents .....	300	WebJavaImageClass Properties .....	349
GetHasHotkey .....	301	Filename .....	349
WebHtmlRegionClass Methods .....	302	WebJavaListClass Methods .....	362
CreateHtmlExtra .....	302	CreateHtml .....	362
SetParentDefaults .....	303	CreateJsldata .....	364
WebHtmlSheetClass Methods .....	305	CreateParams .....	365
CreateHtml .....	305	GetAppletType .....	366
CreateTabControl .....	307	GetEventAction .....	367
GetIsChild .....	308	Init .....	368
GetTableAttributes .....	309	Kill .....	369
Init .....	310	ResetControl .....	370
ResetControl .....	311	ResetFromQueue .....	371
SetBorderWidth .....	312	SetAutoSpotLink .....	372
SetChildDefaults .....	313	SetDirty .....	373

SetEventAction .....	373	Server .....	446
SetQueue .....	375	WebServerClass .....	63
UpdateState .....	376	WebServerClass Methods .....	72
WebJavaListClass Properties .....	359	Connect .....	74
AutoSpotLink .....	359	GetInternetEnabled .....	75
EventActionQ .....	359	GetReadyForPage .....	76
Format .....	360	GetRequestedWholePage .....	77
FromQ .....	360	GetSendWholePage .....	78
QueueActionQ .....	361	Halt .....	79
Started .....	361	Init .....	80
WebJavaStringClass Methods .....	381	Kill .....	81
CreateHtml .....	381	Quit .....	82
CreateJsldata .....	382	SetDialogPageBackground .....	83
CreateParams .....	383	SetDialogWinBackground .....	84
GetAppletType .....	384	SetNewPageDisable .....	86
Init .....	385	SetNextAction .....	87
SetAutoSpotLink .....	386	SetSendWholePage .....	85
WebJavaStringClass Properties .....	380	TakeEvent .....	88
AutoSpotLink .....	380	TakePageSent .....	89
LastText .....	380	WebServerClass Properties .....	66
WebJavaToggleButtonClass Methods .....	343	Active .....	66
GetEventAction .....	343	ArgIndex .....	66
WebListClass .....	355	Broker .....	66
WebListClass Methods .....	355	Client .....	67
GetBackgroundColor .....	400	CommandLine .....	68
WebLiteralClass Methods .....	400	CurSubmit .....	67
CreateHtml .....	400	DialogPageBackColor .....	68
GetCellAttributes .....	401	DialogPageImage .....	68
WebLiteralClass Properties .....	399	DialogWinBackColor .....	68
Text .....	399	DialogWinImage .....	69
WebMenuBarClass Methods .....	429	Files .....	69
CreateHtml .....	429	GotCommandLine .....	69
GetPosition .....	430	JavaLibraryPath .....	70
GetVisible .....	431	PageToReturnTo .....	70
Init .....	432	ProgramName .....	71
WebNullControlClass Methods .....	402	TimeOut .....	71
CreateHtml .....	402	WebServerClass synopsis .....	35
CreateJsldata .....	402	WebStringClass Methods .....	377
GetAppletType .....	403	SetBreakable .....	377
GetCellAttributes .....	404	WebToolBarClass Methods .....	433
GetIsChild .....	405	CreateHtml .....	433
GetVisible .....	406	GetAppletType .....	434
WebReportClass .....	443	GetPosition .....	435
WebReportClass Methods .....	447	GetVisible .....	436
Init .....	447	WebWindowBaseClass Methods .....	153
Kill .....	447	CreateChildHtml .....	153
Preview .....	448	GetBackgroundColor .....	154
SetNumPages .....	448	GetBackgroundImage .....	154
WebReportClass Properties .....	445	GetChildren .....	156
Html .....	445	GetControl .....	155
NumPages .....	446	GetCreateClose .....	155
Q .....	445	GetFirstChild .....	157

GetHelpHandler .....	157	CreateJsldata .....	175
GetHelpReference .....	158	CreatePageFooter .....	176
GetHelpTarget .....	158	CreatePageHeader .....	176
GetMenubarFeq .....	158	CreateUnauthorizedPage .....	177
GetPagelImage .....	159	GetAuthorized .....	179
GetShowMenubar .....	159	GetBackgroundColor .....	180
GetShowToolbar .....	159	GetBackgroundImage .....	181
GetToolbarFeq .....	160	GetButtonInClientArea .....	182
GetToolbarMode .....	160	GetChildren .....	183
GetWebActiveFrame .....	161	GetControl .....	184
WebWindowBaseClass Properties .....	140	GetControlInfo .....	186
AllowJava .....	140	GetCreateClose .....	188
Background .....	140	GetFirstChild .....	187
BackImage .....	140	GetHelpHandler .....	189
BorderWidth .....	141	GetHelpReference .....	190
CloselImage .....	141	GetHelpTarget .....	192
CreateCaption .....	141	GetMenubarFeq .....	191
CreateClose .....	142	GetPagelImage .....	192
CreateToolbar .....	142	GetShowMenubar .....	194
DefaultButton .....	143	GetShowToolbar .....	194
DefaultButtonNeeded .....	143	GetTableAttributes .....	195
DisabledAction .....	144	GetTargetSecurity .....	195
Files .....	144	GetToolbarFeq .....	196
FormatBorderWidth .....	145	GetToolbarMode .....	199
GroupBorderWidth .....	145	GetWebActiveFrame .....	198
HelpDocument .....	145	Init .....	200
HelpEnabled .....	146	Kill .....	200, 201
HelpRelative .....	146	ResetFromControls .....	200
HelpStyle .....	146	SetBackground .....	201
HtmlOption .....	147	SetCentered .....	202
IsSplash .....	147	SetChildDefaults .....	202
MenubarFeq .....	148	SetFormatOptions .....	203
MenubarType .....	148	SetHelpDocument .....	204
OptionBorderWidth .....	148	SetHelpURL .....	205
PageBackground .....	149	SetPageBackground .....	206
PagelImage .....	149	SetPassword .....	207
Server .....	149	SetSplash .....	208
SheetBorderWidth .....	150	SetTimer .....	209
SnapX .....	150	SuppressControl .....	210
SnapY .....	150	TakeCreatePage .....	211
TimerAction .....	151	TakeEvent .....	212
TimerDelay .....	151	TakeRequest .....	213
ToolbarFeq .....	152	TakeUnknownSubmit .....	214
WebWindowClass .....	131	TitleContents .....	215
WebWindowClass Methods .....	165	ValidatePassword .....	216
AddControl .....	167	WebWindowClass Properties .....	162
AddControlsToLayout .....	169	Authorize .....	162
BodyFooter .....	170	AuthorizeArea .....	163
BodyHeader .....	171	HtmlTarget .....	163
CreateChildHtml .....	172	IsCentered .....	163
CreateDummyHtmlPage .....	173	IsSecure .....	164
CreateHtmlPage .....	174	SentHtml .....	164

WebWindowClass synopsis .....	35	HtmlClass Methods .....	564
Width		WriteSubmitApplet	
WebCloseButtonClass Properties .....	387	HtmlClass Methods .....	565
Write		WriteTableFooter	
TextOutputClass Methods .....	577	HtmlClass Methods .....	566
WriteAppletDimParameter		WriteTableHeader	
HtmlClass Methods .....	545	HtmlClass Methods .....	566
WriteAppletFilenameParameter		WriteTableNewCol	
HtmlClass Methods .....	546	HtmlClass Methods .....	567
WriteAppletFontParameter		WriteTableNewRow	
HtmlClass Methods .....	547	HtmlClass Methods .....	567
WriteAppletFooter		WriteText	
HtmlClass Methods .....	548	HtmlClass Methods .....	568
WriteAppletHeader		<b>X</b>	
HtmlClass Methods .....	549	X	
WriteAppletOptParameter		WebCloseButtonClass Properties .....	387
HtmlClass Methods .....	550	<b>Y</b>	
WriteAppletParameter		Y	
HtmlClass Methods .....	551	WebCloseButtonClass Properties .....	387
WriteAppletUAID			
HtmlClass Methods .....	552		
WriteChildAppletFooter			
HtmlClass Methods .....	553		
WriteChildAppletHeader			
HtmlClass Methods .....	554		
WriteContainerAppletHeader			
HtmlClass Methods .....	555		
WriteControlFooter			
HtmlClass Methods .....	556		
WriteControlHeader			
HtmlClass Methods .....	556		
WriteEventHandler			
HtmlClass Methods .....	557		
WriteFontFooter			
HtmlClass Methods .....	559		
WriteFontHeader			
HtmlClass Methods .....	559		
WriteFormFooter			
HtmlClass Methods .....	560		
WriteFormHeader			
HtmlClass Methods .....	560		
WriteJavaScript			
HtmlClass Methods .....	561		
WriteIn			
TextOutputClass Methods .....	578		
WriteOnFocusHandler			
HtmlClass Methods .....	562		
WritePageBody			
HttpPageBaseClass Methods .....	620		
WriteRefreshTimer			
HtmlClass Methods .....	563		
WriteSpace			



---

# ***NOTES***









