

Cadence[®] User Interface SKILL Functions Reference

**Product Version 5.0
December 2002**

© 1990-2002 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-862-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

<u>Preface</u>	23
<u>About This Manual</u>	23
<u>Finding Information in This Manual</u>	23
<u>Other Sources of Information</u>	24
<u>Typographic and Syntax Conventions</u>	24
<u>Data Types</u>	25
<u>SKILL Syntax Examples</u>	26
 <u>1</u>	
<u>Introduction</u>	29
<u>Access to the User Interface</u>	29
<u>Bottom-Up Construction</u>	29
<u>Global Symbols</u>	29
<u>Destroying</u>	30
<u>SKILL Coding</u>	30
<u>Callback Routines</u>	30
<u>Types of Callbacks</u>	30
<u>Restrictions</u>	31
 <u>2</u>	
<u>Command Interpreter Window</u>	33
<u>Overview</u>	33
<u>Mouse/Keyboard Behavior</u>	34
<u>Scrolling</u>	34
<u>Transcript Session</u>	34
<u>Command Design</u>	36
<u>Input Prompt</u>	36
<u>Parse Command Line</u>	36
<u>Command Output Results</u>	36
<u>Error Recovery</u>	37

Cadence User Interface SKILL Functions Reference

<u>Interrupts</u>	37
<u>Nesting Commands</u>	37
<u>General Environmental Functions</u>	38
<u>dplp</u>	39
<u>hiBoxCenter</u>	40
<u>hiEditfile</u>	41
<u>hiFlush</u>	42
<u>hiFocusToCIW</u>	43
<u>hiGetAttention</u>	44
<u>hiGetDisplayName</u>	45
<u>hiGetMultiClickTime</u>	46
<u>hiGetScreenSize</u>	47
<u>hiGraphicMode</u>	48
<u>hiGetBeepVolume</u>	49
<u>hiGetCommandPoint</u>	50
<u>hiGetCIWindow</u>	51
<u>hiGetPoint</u>	52
<u>hiQuit</u>	53
<u>hiRegTimer</u>	54
<u>hiRepeat</u>	55
<u>hiScaleBox</u>	56
<u>hiSetBeepVolume</u>	58
<u>hiSetMultiClickTime</u>	59
<u>hiSetUserPreferences</u>	60
<u>hiSynchronize</u>	61
<u>hiViewfile</u>	62
<u>Process Control Functions</u>	63
<u>hiCheckAbort</u>	64
<u>hiGetUserAbort</u>	65
<u>hiResetAbort</u>	66
<u>hiSetAbort</u>	67
<u>Log File Functions</u>	68
<u>hiEndLog</u>	69
<u>hiGetLogFileName</u>	70
<u>hiIsInReplay</u>	71
<u>hiReplayFile</u>	72

Cadence User Interface SKILL Functions Reference

<u>hiSetFilter</u>	74
<u>hiSetFilterOptions</u>	75
<u>hiStartLog</u>	77
<u>Resource Management Functions</u>	78
<u>hiGetBBoxResource</u>	79
<u>hiGetGeometryResource</u>	80
<u>hiGetStringResource</u>	82
<u>Text Attribute Functions</u>	83
<u>hiGetFont</u>	84
<u>hiGetTextWidth</u>	85
<u>hiQueryFont</u>	86
<u>hiSetFont</u>	87
<u>hiTextWidth</u>	89
<u>Customizing the Environment</u>	90
<u>.cdsenv for Applications</u>	90
<u>Customizing Defaults</u>	91
<u>envCyclicStringToIndex</u>	93
<u>envGetAvailableTools</u>	94
<u>envGetDefVal</u>	95
<u>envGetLoadedTools</u>	96
<u>envGetModifiedTools</u>	97
<u>envGetVal</u>	98
<u>envGetVarType</u>	99
<u>envIsToolModified</u>	100
<u>envIsVal</u>	101
<u>envLoadFile</u>	102
<u>envLoadVals</u>	103
<u>envRegLoadDumpTrigger</u>	104
<u>envRegSetTrigger</u>	105
<u>envSetToolCurrValToDefault</u>	106
<u>envSetToolDefaultToCurrVal</u>	107
<u>envSetVal</u>	108
<u>envSetVarCurrValToDefault</u>	109
<u>envSetVarDefaultToCurrVal</u>	110
<u>envStoreEnv</u>	111

3

Menus	119
Overview	120
Popup Menu	120
Fixed Menu	120
Pulldown Menus	121
Help Menu	121
Menu Functions	122
hiAddMenuItem	123
hiCreate2DMenu	124
hiCreateHorizontalFixedMenu	125
hiCreateMenu	128
hiCreateMenuItem	129
hiCreatePulldownMenu	131
hiCreateSimpleMenu	133
hiCreateSeparatorMenuItem	135
hiCreateSliderMenuItem	136
hiCreateVerticalFixedMenu	138
hiDeleteMenu	140
hiDeleteMenuItem	141
hiDisableMenuItem	142
hiDisplayEdgeMenu	143
hiDisplayFixedMenu	145
hiDisplayMenu	147
hiDisplayWindowMenu	148
hiEdgeFixedMenuDone	149
hiEnableMenuItem	150
hiFixedMenuDown	151
hiGetWindowMenu	152
hiInsertMenuItem	153
hils2DMenu	154
hilsIcon	155
hilsMenu	156
hilsMenuItemEnabled	157
hiSetFixedMenuSize	158

Cadence User Interface SKILL Functions Reference

<u>hiSetMenu</u>	159
<u>hiSetMenuItemCallback</u>	160
<u>hiSetMenuItemText</u>	161
<u>hiSetWindowMenu</u>	162
<u>Programming Samples</u>	163
<u>Creating a Sample Pulldown Menu</u>	164
<u>Creating a Sample Popup Menu</u>	165
<u>Creating a Sample Fixed Menu</u>	166
<u>Creating a Sample Slider Menu</u>	168
<u>Adding an Entry to a Menu</u>	170

4

<u>Dialog Boxes</u>	171
<u>Overview</u>	171
<u>hiDBoxCancel</u>	173
<u>hiDBoxOK</u>	174
<u>hiDisplayAppDBox</u>	175
<u>hiDisplayBlockingDBox</u>	178
<u>hiDisplayModalDBox</u>	179
<u>hiDisplayModelessDBox</u>	180
<u>hiDisplayNonBlockingDBox</u>	181
<u>hiDisplaySysModalDBox</u>	182
<u>Programming Samples</u>	183
<u>Creating a Sample Modeless Dialog Box</u>	183
<u>Creating a Sample Modal Dialog Box</u>	184

5

<u>List Boxes</u>	185
<u>Overview</u>	185
<u>hiDisplayListBox</u>	187
<u>hiGetListBoxFieldFit</u>	188
<u>hiGetNumVisibleItems</u>	190
<u>hiListBoxCancel</u>	191
<u>hiListBoxDone</u>	192
<u>hiShowListBox</u>	193

Cadence User Interface SKILL Functions Reference

<u>Programming Samples</u>	196
<u>Creating a Simple List Box Application</u>	196
<u>Creating an Advanced List Box Application</u>	197

6

Icons

<u>Overview</u>	199
<u>colorIndex</u>	200
<u>hiCreateColorArray</u>	201
<u>hiMatchColor</u>	202
<u>hiMatchColorByName</u>	204
<u>hiStringToIcon</u>	205

7

Forms

<u>Creating and Displaying Forms</u>	208
<u>Standard Forms</u>	208
<u>Options Forms</u>	209
<u>Accessing Form Data</u>	209
<u>Creating Form Fields</u>	210
<u>Form Layout</u>	211
<u>One-Dimensional Form Example</u>	211
<u>Two Dimensional Form Layout</u>	212
<u>Property List Editor</u>	218
<u>Form and Field Functions</u>	219
<u>hiAddCyclicChoice</u>	220
<u>hiAddField</u>	221
<u>hiAddFields</u>	223
<u>hiChangeCyclicChoices</u>	225
<u>hiChangeFormCallback</u>	226
<u>hiChangeFormTitle</u>	227
<u>hiCreateAppForm</u>	228
<u>hiCreateBBoxField</u>	236
<u>hiCreateBooleanButton</u>	239
<u>hiCreateButton</u>	241

Cadence User Interface SKILL Functions Reference

<u>hiCreateButtonBoxField</u>	243
<u>hiCreateComboField</u>	245
<u>hiCreateCyclicField</u>	248
<u>hiCreateFloatField</u>	251
<u>hiCreateFrameField</u>	255
<u>hiCreateScrollRegion</u>	257
<u>hillsScrollRegion</u>	262
<u>hiCreateForm</u>	263
<u>hiCreateFormButton</u>	264
<u>hiCreateFormLabel</u>	266
<u>hiCreateIntField</u>	267
<u>hiCreateLabel</u>	271
<u>hiCreateLayerCyclicField</u>	273
<u>hiCreateListField</u>	275
<u>hiCreateListBoxField</u>	278
<u>hiCreateMLTextField</u>	283
<u>hiCreateOptionsForm</u>	286
<u>hiCreatePointField</u>	287
<u>hiCreatePointListField</u>	290
<u>hiCreateRadioField</u>	293
<u>hiCreateReportField</u>	296
<u>hiCreateScaleField</u>	304
<u>hiCreateSeparatorField</u>	307
<u>hiCreateSpinBox</u>	309
<u>hiCreateStringField</u>	312
<u>hiCreateTabField</u>	316
<u>hiCreateToggleField</u>	319
<u>hiDeleteField</u>	322
<u>hiDeleteFields</u>	323
<u>hiDeleteForm</u>	324
<u>hiDisplayForm</u>	325
<u>hiEditPropList</u>	327
<u>hiEscapeStringChars</u>	330
<u>hiFormApply</u>	331
<u>hiFormCancel</u>	332
<u>hiFormClose</u>	333

Cadence User Interface SKILL Functions Reference

<u>hiFormDefaults</u>	334
<u>hiFormDone</u>	335
<u>hiFormFinish</u>	336
<u>hiFormList</u>	337
<u>hiFormUnmap</u>	338
<u>hiGetCurrentField</u>	340
<u>hiGetFieldInfo</u>	341
<u>hiGetFieldOverlaps</u>	342
<u>hiGetCurrentForm</u>	343
<u>hiGetInsertionPosition</u>	344
<u>hiGetLayerCyclicValue</u>	345
<u>hiGetListBoxValue</u>	346
<u>hiGetScrollBarInfo</u>	347
<u>hiGetTextFieldFit</u>	349
<u>hiGetTopListItem</u>	351
<u>hiHighlightField</u>	352
<u>hiIgnoreProp</u>	354
<u>hiInFormApply</u>	355
<u>hiInstantiateForm</u>	356
<u>hilsForm</u>	357
<u>hilsFormDisplayed</u>	358
<u>hilsInstantiated</u>	359
<u>hilsInFieldCancel</u>	360
<u>hiLayerMatchCyclicStr</u>	361
<u>hiLayerStringToLPP</u>	362
<u>hiMoveField</u>	363
<u>hiMoveInsBarToEnd</u>	364
<u>hiOffsetField</u>	365
<u>hiOffsetFields</u>	367
<u>hiResizeField</u>	369
<u>hiReattachField</u>	370
<u>hiReportSelectItem</u>	372
<u>hiReportSelectItems</u>	374
<u>hiReportSelectAllItems</u>	376
<u>hiReportDeselectItem</u>	377
<u>hiReportDeselectItems</u>	379

Cadence User Interface SKILL Functions Reference

<u>hiReportDeselectAllItems</u>	381
<u>hiReportGetSelectedItems</u>	382
<u>hiSetButtonLabel</u>	383
<u>hiSetCallbackStatus</u>	384
<u>hiSetCurrentField</u>	385
<u>hiSetFieldEditable</u>	387
<u>hiSetFieldEnabled</u>	388
<u>hiSetFormButtonEnabled</u>	389
<u>hiSetFormBlock</u>	390
<u>hiSetFormHighlights</u>	391
<u>hiSetFormMinMaxSize</u>	392
<u>hiSetFormName</u>	393
<u>hiSetFormPosition</u>	394
<u>hiSetFormSize</u>	395
<u>hiSetFormToDefaults</u>	396
<u>hiSetInsertionPosition</u>	397
<u>hiSetLayerCyclicValue</u>	398
<u>hiSetScrollBarValue</u>	399
<u>hiSetTopListItem</u>	401
<u>hiSetListItemVisible</u>	402
<u>hiSetListItemCenter</u>	403
<u>hiShowFieldBorders</u>	404
<u>hiStoreFormLocation</u>	405
<u>hiUpdateFormBlock</u>	406
<u>Programming Samples</u>	407
<u>File Form: Creating String Fields</u>	407
<u>Form Fields: Creating Fields in a Form</u>	409
<u>Window Form: Creating an Application</u>	411
<u>List Editor: Creating List Box Fields</u>	414
<u>File Browser: Creating Multiline Text Fields</u>	418

8

<u>Windows</u>	421
<u>Window Management</u>	422
<u>Window Functions</u>	424
<u>getCurrentWindow</u>	425
<u>getMaxScreenCoords</u>	426
<u>hiGetDbuPoint</u>	427
<u>hiGetDrawThruDelta</u>	428
<u>hiGetScreenPoint</u>	429
<u>hiOpenWindow</u>	430
<u>hiCreateWindow</u>	434
<u>hiDisplayWindow</u>	437
<u>hiGetHelp</u>	438
<u>window</u>	439
<u>wtypep</u>	440
<u>windowp</u>	441
<u>hilsWindowSpecifier</u>	442
<u>hiSetWindowAtts</u>	443
<u>hiSetWinStyle</u>	445
<u>hiCloseWindow</u>	446
<u>hiRegCloseProc</u>	447
<u>hiUnregCloseProc</u>	448
<u>hiGetWindowState</u>	449
<u>hiSwitchWindowType</u>	450
<u>hilconifyWindow</u>	452
<u>hiDeiconifyWindow</u>	453
<u>hiGetWindowIconifyState</u>	454
<u>hiLowerWindow</u>	455
<u>hiRaiseWindow</u>	456
<u>hiPickWindow</u>	457
<u>hiMapWindow</u>	458
<u>hiUnmapWindow</u>	459
<u>hiMoveWindow</u>	460
<u>hiResizeWindow</u>	461
<u>hiFocusToCursor</u>	462

Cadence User Interface SKILL Functions Reference

<u>hiGetWindowList</u>	463
<u>hiGetWindowName</u>	464
<u>hiSetWindowName</u>	465
<u>hiGetIconName</u>	466
<u>hiSetIconName</u>	467
<u>hiSetWindowIcon</u>	468
<u>hiGetWidgetType</u>	469
<u>hiIsWidgetType</u>	470
<u>hiGetAppType</u>	471
<u>hiGetCurrentWindow</u>	472
<u>hiGetMaxScreenCoords</u>	473
<u>hiGetAbsWindowScreenBBox</u>	474
<u>hiGetWMOffsets</u>	475
<u>hiAddFixedMenu</u>	477
<u>hiRemoveFixedMenu</u>	479
<u>hiMoveFixedMenu</u>	480
<u>hiGetWindowFixedMenu</u>	481
<u>hiSetCursor</u>	482
<u>hiGetCursor</u>	484
<u>hiSetCurrentWindow</u>	485
<u>setCurrentWindow</u>	486
<u>hiSetShadowMode</u>	487
<u>hiSetShadowPercent</u>	488
<u>Viewing Functions</u>	489
<u>hiZoomIn</u>	490
<u>hiZoomOut</u>	491
<u>hiZoomRelativeScale</u>	492
<u>hiZoomAbsoluteScale</u>	493
<u>hiGetViewBBox</u>	494
<u>hiPan</u>	495
<u>hiVectorPan</u>	496
<u>hiDeltaPan</u>	497
<u>hiAbsolutePan</u>	498
<u>hiRedraw</u>	499
<u>hiSaveView</u>	500
<u>hiListView</u>	501

Cadence User Interface SKILL Functions Reference

<u>hiRestoreView</u>	502
<u>hiPrevWinView</u>	503
<u>hiNextWinView</u>	504
<u>hiGetUndoLimit</u>	505
<u>hiSetUndoLimit</u>	506
<u>hiUndo</u>	507
<u>hiRedo</u>	508
<u>Splash Screen Functions</u>	509
<u>hiAbout</u>	510
<u>hiSetSplashBackground</u>	511
<u>hiSetSplashDefaultBackground</u>	512
<u>hiSetSplashIcon</u>	513
<u>hiSetSplashFamily</u>	514
<u>hiSetSplashProduct</u>	515
<u>hiSetSplashLicense</u>	516

9

Window Banner Functions 517

<u>Overview</u>	517
<u>hiChangeBannerLabel</u>	519
<u>hiDeleteBannerLabel</u>	520
<u>hiInsertBannerMenu</u>	521
<u>hiDeleteBannerMenu</u>	523
<u>hiDeleteBannerMenus</u>	524
<u>hiDeleteStatusBanner</u>	525
<u>hiGetBannerMenus</u>	526
<u>hiGetNumMenus</u>	527
<u>hiIsMenuSlotFilled</u>	528
<u>hiReplaceAllBannerMenus</u>	529

10

Bindkeys 531

<u>Overview</u>	532
<u>hiSetBindKey</u>	533
<u>hiSetBindKeys</u>	538

Cadence User Interface SKILL Functions Reference

<u>hiGetBindKey</u>	540
<u>hiShowBindKeys</u>	542
<u>hiShowBindKeysByAppType</u>	544
<u>hiShowBindKeysByWindow</u>	546
<u>hiRegisterBindKeyPrefix</u>	547
<u>hiInheritBindKey</u>	548
<u>hiGetBindKeyInheritRoot</u>	549
<u>hiGetBindKeyInheritAlias</u>	550
<u>hiGetBindKeyPrefixList</u>	551

11

User Entry Functions 553

Overview 554

<u>Origin of Enterfunction Data</u>	554
<u>Enterfunction Prompts</u>	555
<u>Option Forms</u>	555
<u>Preloading Points</u>	556
<u>Number of Points To Get</u>	556
<u>Terminating an Enterfunction</u>	556
<u>Callback Procedures</u>	557
<u>Enterfunction Flags</u>	558
<u>Nesting Enterfunctions</u>	559

User Entry Functions 562

<u>enterArc</u>	564
<u>enterBox</u>	567
<u>enterCircle</u>	571
<u>enterDonut</u>	574
<u>enterEllipse</u>	578
<u>enterLine</u>	581
<u>enterNumber</u>	584
<u>enterPath</u>	586
<u>enterPoint</u>	590
<u>enterPoints</u>	593
<u>enterPolygon</u>	596
<u>enterScreenBox</u>	599

Cadence User Interface SKILL Functions Reference

<u>enterSegment</u>	601
<u>enterMultiRep</u>	604
<u>enterString</u>	608
<u>addPoint</u>	610
<u>preXY</u>	612
<u>deletePoint</u>	613
<u>cancelEnterFun</u>	615
<u>finishEnterFun</u>	616
<u>applyEnterFun</u>	617
<u>changeEnterFun</u>	619
<u>changeNextEnterFun</u>	622
<u>clearAllEnterFunctions</u>	623
<u>hiGetCurrentCmd</u>	624
<u>hiInEnterFun</u>	625
<u>hiMarkNestable</u>	626
<u>hiMarkNonNestable</u>	627
<u>hiUpdate</u>	628
<u>hiToggleEnterForm</u>	629
<u>undrawEnterFun</u>	630
<u>drawEnterFun</u>	631

12

Encapsulation Window 633

<u>Overview</u>	634
<u>hiEncap</u>	635
<u>hiSetEncapSkillCmd</u>	637
<u>hiGetEncapSkillCmd</u>	639
<u>hiSetEncapPrompt</u>	640
<u>hiSetEncapHistory</u>	641
<u>hiAppendInputCmd</u>	643
<u>hiFocusToEncap</u>	644

13

<u>Viewfile Window</u>	645
<u>Viewfile Management</u>	646
<u>view</u>	647
<u>hiSetViewfile</u>	649
<u>hiSaveViewfile</u>	650
<u>hiSaveAsViewfile</u>	651
<u>hiStartGenTextIndex</u>	652
<u>hiGenTextIndex</u>	653
<u>hiGetTextSelection</u>	654
<u>hiGetTextSelByLoc</u>	655
<u>hiSetTextSelection</u>	656
<u>hiSelectTextByLoc</u>	657
<u>hiSetTextSelectAll</u>	658
<u>hiUnselectText</u>	659
<u>hiUnselectTextByLoc</u>	660
<u>hiUnselectTextClass</u>	661
<u>hiUnselectTextAll</u>	662
<u>hiScrollWindowLeft</u>	663
<u>hiScrollWindowRight</u>	664
<u>hiScrollWindowUp</u>	665
<u>hiScrollWindowDown</u>	666
<u>hiScrollWindowTop</u>	667
<u>hiScrollWindowBottom</u>	668
<u>hiScrollWindowToCurrentIndex</u>	669
<u>hiScrollWindowToIndex</u>	670
<u>hiGetCurrentIndex</u>	671
<u>hiSetCurrentIndex</u>	672
<u>hiDisableTailViewfile</u>	673
<u>hiEnableTailViewfile</u>	674
<u>hiGetTextClass</u>	675
<u>hiSetTextClass</u>	676
<u>Setting the Highlight Color</u>	677
<u>hiSetTextHighlightColor</u>	678
<u>hiTextDisplayString</u>	680

Cadence User Interface SKILL Functions Reference

<u>hiRefreshTextWindow</u>	681
<u>hiUpdateTextSelectionColors</u>	682
<u>Word Delimiters</u>	683
<u>hiGetTextWordDelimiter</u>	684
<u>hiAddTextWordDelimiter</u>	685
<u>hiRemoveTextWordDelimiter</u>	686
<u>hiReplaceTextWordDelimiter</u>	687
<u>Miscellaneous</u>	688
<u>hiGetTextCharAtLoc</u>	689
<u>hiGetTextSourceLength</u>	690
<u>hiGetTextLineColumn</u>	691
<u>hiGetTextIndexLoc</u>	692
<u>hiGetTextDispLoc</u>	693
<u>hiDisableTextSelDefault</u>	694

A

<u>Display Lists</u>	695
<u>Display List Objects</u>	696
<u>Creating a Display List</u>	697
<u>dlMakeDisplayList</u>	698
<u>dlSetClearOnDraw</u>	699
<u>Adding Objects to the Display List</u>	700
<u>dlAddArc</u>	701
<u>dlAddBox</u>	702
<u>dlAddCircle</u>	703
<u>dlAddDonut</u>	704
<u>dlAddEventObject</u>	705
<u>dlAddPath</u>	707
<u>dlAddPoint</u>	708
<u>dlAddPolygon</u>	709
<u>dlAddRasterText</u>	710
<u>dlAddSegment</u>	712
<u>dlAddSkillObject</u>	713
<u>dlAddStrokeText</u>	714

Cadence User Interface SKILL Functions Reference

<u>Display List Draw Functions</u>	716
<u>dlDrawArc</u>	717
<u>dlDrawBox</u>	718
<u>dlDrawCircle</u>	719
<u>dlDrawDonut</u>	720
<u>dlDrawPath</u>	721
<u>dlDrawPoint</u>	722
<u>dlDrawPolygon</u>	723
<u>dlDrawRasterText</u>	724
<u>dlDrawSegment</u>	726
<u>dlDrawStrokeText</u>	727
<u>Display List Pens</u>	728
<u>dlMakePenTable</u>	729
<u>dlMakeStipple</u>	730
<u>dlSetCurrentPen</u>	731
<u>dlSetPenColor</u>	732
<u>dlSetPenFillStyle</u>	733
<u>dlSetPenFilled</u>	734
<u>dlSetPenStipple</u>	735
<u>dlSetPenTable</u>	736
<u>Viewing a Display List</u>	737
<u>dlAttachDlistToWidget</u>	738
<u>dlAttachDlistToWindow</u>	739
<u>dlClearDisplayList</u>	740
<u>dlCloseWidget</u>	741
<u>dlConfigureButton</u>	742
<u>dlDetachDlistFromWidget</u>	743
<u>dlDetachDlistFromWindow</u>	744
<u>dlDisplay</u>	745
<u>dlDlistToIcon</u>	746
<u>dlEnableItem</u>	747
<u>dlFitDlistOnDraw</u>	748
<u>dlMakeDlistButton</u>	749
<u>dlMakeWidget</u>	750
<u>dlMapWidget</u>	751
<u>dlMoveButton</u>	752

Cadence User Interface SKILL Functions Reference

<u>dlResizeButton</u>	753
<u>dlSaveDlist</u>	754
<u>dlSetDlistPosition</u>	755
<u>dlSetDlistScale</u>	756
<u>dlSetWidgetName</u>	757
<u>dlUnMapWidget</u>	758
<u>A Sample Display List</u>	759

B

<u>Graph Browser</u>	761
<u>Overview</u>	762
<u>Key Terms</u>	764
<u>Properties of the dagArc</u>	766
<u>Properties of the dagClass</u>	767
<u>Properties of the dagNode</u>	769
<u>Properties of the dagTool</u>	772
<u>Actions and Action Lists</u>	777
<u>dag Functions</u>	779
<u>dagAddActionToObject</u>	779
<u>dagCreateClass</u>	780
<u>dagCreateNode</u>	781
<u>dagDeleteActionFromObject</u>	782
<u>dagDestroyNode</u>	783
<u>dagDisplayTool</u>	784
<u>dagGetCurrentObject</u>	785
<u>dagGetCurrentTool</u>	787
<u>dagLinkParentToChild</u>	788
<u>dagNumToTool</u>	789
<u>dagOpenTool</u>	790
<u>dagPopTool</u>	792
<u>dagPushTool</u>	793
<u>dagRefreshObject</u>	794
<u>dagSetActionStatus</u>	795
<u>dagSetCurrentTool</u>	796
<u>dagSetExpandedActionStatus</u>	797

Cadence User Interface SKILL Functions Reference

<u>dagSetFont</u>	798
<u>dagUnlinkParentFromChild</u>	799

C

<u>Interprocess Communication</u>	801
<u>hiActivateBatch</u>	802
<u>hiActivateMessages</u>	802
<u>hiBatchProcess</u>	802
<u>hiBeginProcess</u>	802
<u>hiCloseChild</u>	802
<u>hiContChild</u>	802
<u>hiGetExitStatus</u>	802
<u>hiGetPid</u>	802
<u>hiGetPriority</u>	803
<u>hilsActiveChild</u>	803
<u>hilsAliveChild</u>	803
<u>hiKillAllProcs</u>	803
<u>hiKillChild</u>	803
<u>hiReadChild</u>	803
<u>hiSetPriority</u>	803
<u>hiSetSkillDesc</u>	803
<u>hiSkillProcess</u>	804
<u>hiSleep</u>	804
<u>hiSoftInterrupt</u>	804
<u>hiStopChild</u>	804
<u>hiWait</u>	804
<u>hiWaitForChild</u>	804
<u>hiWriteChild</u>	804

D

<u>Strokes</u>	805
<u>Overview</u>	805
<u>hiGetStrokeBBox</u>	806
<u>hiGetStrokeFirstPt</u>	807
<u>hiGetStrokeLastPt</u>	808

Cadence User Interface SKILL Functions Reference

<u>hiStroke</u>	809
<u>Index</u>	811

Preface

The following topics are discussed in this Preface:

- [About This Manual](#) on page 23
- [Other Sources of Information](#) on page 24
- [Typographic and Syntax Conventions](#) on page 24

About This Manual

The *Cadence User Interface SKILL Functions Reference Manual* contains information about modifying user interfaces programmatically through SKILL functions. This manual assumes that you are familiar with the SKILL programming language.

Finding Information in This Manual

The following table summarizes the topics covered in this manual.

To Find out How to. . .	Read . . .
Access the user interface	Chapter 1, "Introduction"
Access and customize the Command Interpreter window	Chapter 2, "Command Interpreter Window"
Create and modify menus	Chapter 3, "Menus"
Create and modify dialog boxes	Chapter 4, "Dialog Boxes"
Create and modify list boxes	Chapter 5, "List Boxes"
Create and modify icons	Chapter 6, "Icons"
Create, lay out and modify forms	Chapter 7, "Forms"
Create and modify windows	Chapter 8, "Windows"
Create and modify banners	Chapter 9, "Window Banner Functions"
Create and modify bindkeys	Chapter 10, "Bindkeys"

Cadence User Interface SKILL Functions Reference

Preface

To Find out How to. . .

Read . . .

Digitize shapes

Chapter 11, "User Entry Functions"

Create and modify encapsulation windows

Chapter 12, "Encapsulation Window"

Create and modify viewfile windows

Chapter 13, "Viewfile Window"

Construct and display shapes

Appendix A, "Display Lists"

Create a browser tool

Appendix B, "Graph Browser"

Other Sources of Information

For more information about the user interface and SKILL programming, see the following:

- [*SKILL Language User Guide*](#)
- [*SKILL Language Reference Manual*](#)

Typographic and Syntax Conventions

This section describes typographic and syntax conventions used in this manual.

text	Indicates text you must type exactly as it is presented.
<i>z_argument</i>	Indicates text that you must replace with an appropriate argument. The prefix (in this case, <i>z_</i>) indicates the data type the argument can accept. Do not type the data type or underscore.
[]	Denotes optional arguments. When used with vertical bars, they enclose a list of choices from which you can choose one.
{ }	Used with vertical bars and encloses a list of choices from which you must choose one.
	Separates a choice of options; separates the possible values that can be returned by a Cadence® SKILL language function.
...	Indicates that you can repeat the previous argument.

Cadence User Interface SKILL Functions Reference

Preface

=> Precedes the values returned by a Cadence SKILL language function.

text Indicates names of manuals, menu commands, form buttons, and form fields.

Important

The language requires many characters not included in the preceding list. You must type these characters exactly as they are shown in the syntax.

Data Types

The Cadence® SKILL language supports several data types to identify the type of value you can assign to an argument. Data types are identified by a single letter followed by an underscore; for example, *t* is the data type in *t_viewNames*. Data types and the underscore are used as identifiers only: they are not to be typed.

The table below lists all data types supported by SKILL.

Data Types by Type

Prefix	Internal Name	Data Type
<i>a</i>	array	array
<i>b</i>	ddUserType	Boolean
<i>C</i>	opfcontext	OPF context
<i>d</i>	dbobject	Cadence database object (CDBA)
<i>e</i>	envobj	environment
<i>f</i>	flonum	floating-point number
<i>F</i>	opffile	OPF file ID
<i>g</i>	general	any data type
<i>G</i>	gdmSpecIIUserType	gdm spec
<i>h</i>	hdbobject	hierarchical database configuration object
<i>l</i>	list	linked list
<i>m</i>	nmplIUserType	nmplI user type
<i>M</i>	cdsEvalObject	—

Cadence User Interface SKILL Functions Reference

Preface

Data Types by Type, *continued*

Prefix	Internal Name	Data Type
<i>n</i>	number	integer or floating-point number
<i>o</i>	userType	user-defined type (other)
<i>p</i>	port	I/O port
<i>q</i>	gdmspecListIUUserType	gdm spec list
<i>r</i>	defstruct	defstruct
<i>R</i>	rodObj	relative object design (ROD) object
<i>s</i>	symbol	symbol
<i>S</i>	stringSymbol	symbol or character string
<i>t</i>	string	character string (text)
<i>u</i>	function	function object, either the name of a function (symbol) or a lambda function body (list)
<i>U</i>	funobj	function object
<i>v</i>	hdbpath	—
<i>w</i>	wtype	window type
<i>x</i>	integer	integer number
<i>y</i>	binary	binary function
<i>&</i>	pointer	pointer type

SKILL Syntax Examples

The following examples show typical syntax characters used in user interface SKILL.

Example 1

```
list(  
    g_arg1  
    [g_arg2] ...  
)  
  
=> l_result
```

This example illustrates the following syntax characters:

Cadence User Interface SKILL Functions Reference

Preface

<code>list</code>	Plain type indicates words that you must enter literally.
<code><i>g_arg1</i></code>	Words in italics indicate arguments for which you must substitute a name or a value.
<code>()</code>	Parentheses separate names of functions from their arguments.
<code>_</code>	An underscore separates an argument type (left) from an argument name (right).
<code>[]</code>	Brackets indicate that the enclosed argument is optional.
<code>=></code>	A right arrow points to the description of the return value of the function.
<code>...</code>	Three dots indicate that the preceding item can appear any number of times

Example 2

```
needNCells(  
    s_cellType | st_userType  
    x_cellCount  
)  
=> t/nil
```

This example illustrates two additional syntax characters

<code> </code>	Vertical bars separate a choice of required options.
<code>/</code>	Slashes separate possible return values.

Cadence User Interface SKILL Functions Reference

Preface

Introduction

The following topics are discussed in this chapter:

- [Access to the User Interface](#) on page 29
- [Callback Routines](#) on page 30

Access to the User Interface

At system start-up, the look and feel of the system is determined by a configurable system start-up file. Once the system is running, changing the system's look and feel, creating new objects, and object manipulation is possible through SKILL and C user interface access routines. The goal of these interface routines is to provide layered, controlled access to the user interface and its objects.

Access is available through high-level objects, composed of primitive user interface elements: menus, scrollbars, popup menus, labels, command buttons, and so forth, or through primitive objects themselves. Several layers of start-up configuration files are supplied. You can customize the functionality on each level.

Bottom-Up Construction

Larger, composite user-interface objects must be created in a bottom-up fashion; that is, each sub-object that makes up a composite object must be created before the composite object is created. A menu is an example of a composite object. Each menu item must be created before the call to `hiCreateMenu`, which accepts menu item objects as arguments.

Global Symbols

Most of the user interface objects you create (that is, forms, menus, list boxes, non-blocking dialog boxes) are used and accessed in more than one SKILL routine. For this reason it is necessary to use a unique global SKILL symbol as the SKILL handle in the creation routines. The user interface code depends on this.

Cadence User Interface SKILL Functions Reference

Introduction

For example, when the form is created and the SKILL handle passed to `hiCreateAppForm` or `hiCreateForm` is not global, any attempts to bring this form down (via the *OK* or *Cancel* button) will fail. This is because the user interface code cannot access the form contents if the form symbol is not global.

Destroying

Do not try to destroy a user-interface object by resetting its SKILL handle to `nil`. This will destroy the SKILL contents of the object, but will leave the memory associated with the object unreferenced, and impossible to free. For menus and forms, use the appropriate delete function (that is, `hiDeleteMenu` or `hiDeleteForm`). List boxes and dialog boxes will be properly destroyed when they are removed from the screen.

SKILL Coding

General information about programming in SKILL is available in the *SKILL Language User Guide* and the *SKILL Language Reference Manual*. Understand the coding techniques described in these documents before accessing the user interface.

Callback Routines

User interface objects can have callback routines associated with them. In this case, a callback is a SKILL routine that is asynchronously “called back” if some triggering user action takes place, for example, in response to values being changed on forms, menu item selection, or dialog box confirmation.

Types of Callbacks

SKILL callbacks can be strings or symbols. In either case, the string or function specified by its symbol is evaluated in SKILL at the time the callback is triggered.

If the callback is a string, it must be a complete, valid SKILL expression, as this string is passed directly to SKILL for evaluation. If the callback is a symbol, it is assumed that the value of this symbol is a function, and a predetermined set of arguments are passed to that function.

Typically, callbacks analyze and process data that you provide. In the case of forms, for example, a callback can be specified for each form field or for the form itself.

Restrictions

There are a few restrictions on the functionality of callback routines:

- They cannot return data on the stack. Instead, callback expressions that return a value might do so in this manner:

```
someSymbol = hiSomeRoutine(y)
```

In this example, `someSymbol` is a global SKILL symbol to be accessed later.

- Make callbacks as short and as clean as possible. In particular, be careful to avoid corrupting program data. As an example, the results can be unpredictable if a form callback modifies the form itself. Callbacks with destructive actions such as form modification should be associated with actions that cannot be reversed, and cannot be done programmatically; that is, do not have a destructive action associated with a toggle button press (the toggle button has two states). Because this process can be simulated programmatically, doing so may unintentionally execute the destructive callback multiple times. In this case, the callback should be associated with a form button, which is an irreversible, non-programmatic action.
- Avoid recursion in programming callbacks. This is a common mistake, and is easily demonstrated by a field callback which again sets the value of the field itself. Since callbacks are called when the value of the field changes, this would recursively continue calling the field callback, which sets the field value, which calls the field's callback, which sets the field's value, and so on.

If an error is detected in a callback routine, a dialog box should be brought up to flag the error, or an error logged in the CIW output panel. The field (type-in fields only) can be highlighted as well.

Cadence User Interface SKILL Functions Reference

Introduction

Command Interpreter Window

The following topics are discussed in this chapter:

- [Overview](#) on page 33
- [General Environmental Functions](#) on page 38
- [Process Control Functions](#) on page 63
- [Log File Functions](#) on page 68
- [Resource Management Functions](#) on page 78
- [Text Attribute Functions](#) on page 83
- [Customizing the Environment](#) on page 90

Overview

The Command Interpreter Window (CIW) is the first window you see and interact with using Cadence software. It contains menus with commands that allow you to do any operation Cadence supports. It is composed of different areas with separate functions, which are

- Window manager (optional) border
- Output history area
- Prompt line
- Input line
- Mouse bindings line

You control the size and location of the CIW. It can stay on your screen throughout your work session and be temporarily hidden by other windows.

The optional window manager border contains the location of the transcript log file that is kept for each session. The banner contains the pulldown menus.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Each command line is copied into the output area from the input line as it is accepted for execution. Any error messages or output resulting from a command may appear at the bottom of the output area, depending on the settings of the log filter flags. This area is treated as a read-only disk file. The contents cannot be changed by direct keyboard entry.

The input line holds the text of the current prompt until it is ready to be input. Input methods are treated later in this document. For now, it is assumed that command line text accumulates in this command input area, and a single event, usually typing a carriage return, triggers command execution.

Mouse/Keyboard Behavior

The CIW is activated by your keystrokes or mouse actions, just as the other windows are. All function keys, control keys, and mouse buttons are user configurable in any window. All alphanumeric or symbol character keystrokes will have their literal meaning. It is a good idea to avoid binding any alphanumeric or symbol character keys to a command in the CIW.

The interpretation of the keyboard and mouse actions depends on the application type of the window. All keys (including function and control keys and mouse buttons) are user-settable in any window. Pressing a key submits a user-defined SKILL command string to the CIW for evaluation.

Scrolling

The CIW output area contains a vertical scroll bar one column wide. The scroll bar indicates if a part of the window is not displaying.

You can use vertical scrolling in the command output area to examine the history of the session. You can view the last 250 lines of the log in the command output area.

If you enter a new command, or the log receives an error message, the scroll is automatically repositioned to show the end of the log, that is, the last command you typed or the error message received.

Transcript Session

This section describes the transcript log requirements and the contents of the transcript log.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Log Requirements

A transcript log for the current session, called `CDS.log`, is written to disk and a filtered subset of the information is displayed in the CIW output area. The `CDS.log` file provides you with recorded feedback during the session. The information displayed is configurable using the settings in the *Log Filter* command on the Set Options menu. These settings affect what is displayed on the CIW, not what's written to the file.

Log Contents

Anything you type on the input line is composed in the CIW input window and written into the transcript log. Recording the information in the transcript log simplifies additional processing, because it can be copied from the log and pasted on the input line. Another use of the transcript log file is to replay a session to get the system (or cellview) back to a previously known state, which is particularly useful after a crash. Other uses of the log file include regression testing, demonstration scripts, and analysis of system/command throughput.

The SKILL command interpreter is a read, evaluate, and print loop. All commands that execute successfully produce output, even if the output is only the symbol `t`.

All output is written to the session log and labelled with the following log types:

<code>\o</code>	program output
<code>\p</code>	prompt output
<code>\t</code>	typed output
<code>\r</code>	accelerated output
<code>\w</code>	warning output
<code>\e</code>	error output
<code>\i</code>	typed input
<code>\a</code>	accelerated input

Normally, all SKILL commands are implemented as proper functions. When the current window changes, the corresponding SKILL command is written to the log file.

You can use parameters that are sent to SKILL data structures as input to some other command. A few commands write data directly to the transcript log and, by implication, to the output area.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Error messages can originate in a variety of ways. You do not need to distinguish between syntax errors and other kinds of fatal errors. Error messages cannot cause a pause on replay.

Command Design

No SKILL command behaves as if it alone has control of the screen. The origin of a command, by keyboard, menu selection, mouse button action, or replay from a session log, is independent of the command's implementation. The command must be designed to execute regardless of the system's state, mode, or context, except in well-documented cases. The command interpreter must be able to carry out a well-defined sequence of actions when all system initialization is complete.

Input Prompt

When system initialization is complete, SKILL sends a prompt message to the prompt line and waits for an input event to occur. You can choose a different prompting style through the SKILL function `setPrompts`. If you do choose to change styles, the SKILL expression is evaluated for each prompt. SKILL remains inactive until the command line message is received.

Parse Command Line

When a complete command line is received, it is parsed by SKILL. Arguments for SKILL functions are type checked against a template. All reasonable checking must be done at this level in the interests of consistency and economy of coding by the individual functions. All functions are executed according to the precedence of the SKILL command language.

Command Output Results

When a command line is received, SKILL sends the command language message to the output subwindow and to the transcript log. The result of the command is also written to the transcript log and possibly to the command output subwindow, depending on the setting of the log filter flags.

When a command completes successfully, the result is printed to the transcript log (and possibly to the output subwindow).

Error Recovery

If an error is detected while evaluating a SKILL command, the standard error handler formats and outputs a message before returning to the top level. This behavior can be modified by the use of `errSet` and `errSetString` within your SKILL program.

If you have purchased the SKILL Development product, the standard error handler prints the contents of the SKILL stack and leaves you in a nested command interpreter to allow debugging. This includes detailed examination of the stack, and printing and modification of local variables. This nested interpreter runs using the same CIW windows as the top-level command interpreter. It is possible to switch between this and the nondevelopment error handler.

If either of the standard error handlers are inappropriate, you can define your own with the SKILL Development product.

Interrupts

Interrupts are treated as errors. The execution resumes after an interrupt if you use the default error handler. The exception to this rule is `Control-C`, which always terminates the execution of the current SKILL command.

Nesting Commands

You can type a command in the CIW that prompts for more data, type another command that supports nesting, and then complete your first command. With command nesting you can gather additional data; the system supports multiple levels of command nesting. The default is one level, with a user-definable maximum of three levels of nesting.

General Environmental Functions

This section describes functions related to the general Cadence environment and the CIW. Functions are listed alphabetically.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

dplp

```
dplp(  
    g_arg  
)  
=> t / nil
```

Description

This predicate function returns `t` if the argument passes a test to determine whether it is a disembodied property list, and returns `nil` otherwise.

The argument passes the test if it is a list that has an odd number of members and the first list member is `nil`.

A disembodied property list (DPL) is a list of symbol/value pairs that is not attached to a specific object or symbol.

Arguments

<i>g_arg</i>	Data to be evaluated.
--------------	-----------------------

Value Returned

<code>t</code>	Meets the test for a disembodied property list.
----------------	---

<code>nil</code>	Is not a disembodied property list.
------------------	-------------------------------------

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiBoxCenter

```
hiBoxCenter(  
    l_boundingBox  
)  
=> l_boxCenter
```

Description

Returns the center point of a bounding box.

Arguments

<i>l_boundingBox</i>	Bounding box. A valid bounding box is represented by a list of two points separated by blank space. Each point is a list of two integers: the first point is the lower-left coordinate of the box, the second point is the upper-right coordinate.
----------------------	--

Value Returned

<i>l_boxCenter</i>	List of two integers representing the center point of the bounding box.
--------------------	---

Reference

[hiScaleBox](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiEditfile

```
hiEditfile( )  
=> t / nil
```

Description

Brings up a form in which you can specify the file you want to edit and then calls the `edit` function on the file.

Arguments

None.

Value Returned

`t` Returns `t` after the form is closed if you clicked `OK` on the form.

`nil` Returns `nil` after the form is closed if you clicked `Cancel` on the form.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiFlush

```
hiFlush( )  
=> t
```

Description

Causes event queues to be synchronized and pending events to be discarded if the pending event queue is too full. This avoids having the socket forced closed. Used primarily for quicktests.

Arguments

None.

Value Returned

t Returns t when the queue is flushed.

Reference

[hiSynchronize](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiFocusToCIW

```
hiFocusToCIW(  
    [w_windowId]  
)  
=> t / nil
```

Description

Temporarily refocuses the input window to the CIW until a return is entered.

The input focus is then refocused back to the window specified by *w_windowId*.

Arguments

<i>w_windowId</i>	Window where you want the function to act. If the <i>w_windowId</i> is not supplied, the function acts on the current window.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the input window is refocused.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the input window is not refocused.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetAttention

```
hiGetAttention(  
    [x_loudness]  
)  
=> t / nil
```

Description

Rings the bell in the keyboard or terminal.

Arguments

<i>x_loudness</i>	Overrides the loudness of the beep for that iteration. <i>x_loudness</i> ranges from -100 (silence) to 100 (full volume). The default is zero. Setting <i>x_loudness</i> to zero causes the beep to be at the system default loudness. The absolute value of a negative number is the percentage from the system default loudness toward silence, and a positive number is the percentage from the system default loudness toward full volume. The amount of loudness control is system dependent. For example, Sun systems have the same volume for values -99 through 100, but HP systems have a wide range of volume.
-------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if successful.
<i>nil</i>	Returns <i>nil</i> and issues an error message if not successful.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetDisplayName

```
hiGetDisplayName( )  
=> t_name / nil
```

Description

Returns the current display name.

In non-graphical mode, the display name will normally be *<hostname>:500.1*. In graphical mode, the display name will usually be *:0.0* when displaying locally or *<hostname>:0.0* when displaying remotely.

Arguments

None.

Value Returned

<i>t_name</i>	Current display name.
---------------	-----------------------

Reference

[hiGetScreenSize](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetMultiClickTime

```
hiGetMultiClickTime( )  
=> x_milliseconds
```

Description

Returns the number of milliseconds that must elapse before a mouse click is recognized as a separate single click rather than the second click of a double click.

Arguments

None.

Value Returned

<i>x_milliseconds</i>	Number of milliseconds that must elapse before a mouse click as recognized as a separate single click.
-----------------------	--

Reference

[hiSetMultiClickTime](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetScreenSize

```
hiGetScreenSize( )  
=> l_sizes( l_widthHeightPixels l_widthHeightMillimeters )
```

Description

Returns the size of the current display screen as a list of two lists, the first being the width and height in pixels, the second being the width and height in millimeters.

Arguments

None.

Value Returned

<i>l_sizes</i>	The size of the current display screen, expressed as a list of two lists.
----------------	---

<i>l_widthHeightPixels</i>	Width and height in pixels
----------------------------	----------------------------

<i>l_widthHeightMillimeters</i>	Width and height in millimeters
---------------------------------	---------------------------------

Reference

[hiGetDisplayName](#)

hiGraphicMode

```
hiGraphicMode( )  
=> t / nil
```

Description

Returns `t` if the Design Framework II environment is currently running in graphical mode, `nil` if it is running in `-nograph` mode.

Arguments

None.

Value Returned

`t` Returns `t` if in graphical mode.

`nil` Returns `nil` if in non-graphical mode.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetBeepVolume

```
hiGetBeepVolume(  
    )  
=> x_loudness
```

Description

Retrieves the loudness of the beep.

Arguments

None.

Value Returned

x_loudness

An integer indicating the loudness of the beep. It ranges from -100 (silence) to 100 (full volume). The default is zero. Zero means the beep is at the system default loudness. The absolute value of a negative number is the percentage from the system default loudness toward silence, and a positive number is the percentage from the system default loudness toward full volume. The amount of loudness control is system dependent. For example, Sun systems have the same volume for values -99 through 100, but HP systems have a wide range of volume.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetCommandPoint

```
hiGetCommandPoint(  
    [w_windowId]  
)  
=> pointList / nil
```

Description

Gets the location where the cursor was the last time a mouse button or keyboard key was pressed, but only until the cursor leaves that window.

Arguments

<i>w_windowId</i>	Window where you want the function to act. If <i>w_windowId</i> is not specified, the function defaults to the current window.
-------------------	--

Value Returned

<i>pointList</i>	Returns the <i>pointList</i> (x:y position) in user units.
<i>nil</i>	Returns <i>nil</i> if the <i>w_windowId</i> is invalid or if the cursor is not in the specified window.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetCIWindow

```
hiGetCIWindow(  
    )  
=> w_windowId
```

Description

Retrieves the window identity of the CIW.

Arguments

None.

Value Returned

<i>w_windowId</i>	Returns the window identity of the Command Interpreter Window.
-------------------	--

Example

```
hiGetCIWindow( )  
=> window:1
```

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetPoint

```
hiGetPoint(  
    w_windowId  
)  
=> pointList / nil
```

Description

Retrieves the xy position for the current location of the cursor within the boundaries of the window. It returns the xy position even if the cursor is in another window or form that is covering the specified window, as long as the cursor is within the boundaries of the specified window.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
-------------------	--

Value Returned

<i>pointList</i>	Returns the <i>pointList</i> xy position in user units.
<i>nil</i>	Returns <i>nil</i> if the <i>w_windowId</i> is invalid, or if the cursor is outside the boundaries of the specified widow.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiQuit

`hiQuit()`

Description

Quits the Design Framework II session.

`hiQuit` first displays a dialog box that asks *“Do you really wish to quit?”* and then calls `exit` if you click Yes.

When typing in the CIW, it is quicker to use `exit`. When assigning bindkeys, it is safer to use `hiQuit`, since you cannot immediately quit by accidentally pressing the bindkey.

Arguments

None.

Value Returned

`nil` Returns `nil` if `exit` failed.

Note: This function does not return a value if `exit` succeeds.

Reference

[exit](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiRegTimer

```
hiRegTimer(  
    t_callbackString  
    x_tenthsOfSeconds  
)  
=> t / nil
```

Description

Registers a SKILL function string that is executed after the specified time.

Arguments

<i>t_callbackString</i>	SKILL function name (string) that is to be executed after the specified time (<i>x_tenthsOfSeconds</i>) has passed.
<i>x_tenthsOfSeconds</i>	Time, in tenths of seconds, after which the callback is executed.

Value Returned

<i>t</i>	Returns <i>t</i> if successful.
<i>nil</i>	Returns <i>nil</i> if unsuccessful.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiRepeat

```
hiRepeat(  
    )  
=> t / nil
```

Description

Re-executes the last command processed by the CIW.

Arguments

None.

Value Returned

t Returns t if the command is reexecuted.

nil Returns nil and issues an error message if the command is not reexecuted.

hiScaleBox

```
hiScaleBox(  
    l_boundingBox  
    n_scale  
)  
=> l_scaledBBox
```

Description

Returns a bounding box that has the same center point as the bounding box passed in, but whose sides are scaled by $1/n_scale$.

For example, if you pass 2 as the scale, the sides are half as long. If you pass 0.5 as the scale, the sides are twice as long.

You enter the lower left and upper right coordinates of the bounding box. The function calculates the center point, scales the sides, and returns the new lower left and upper right coordinates.

Arguments

<i>l_boundingBox</i>	List of the lower left and upper right coordinates of the initial bounding box.
<i>n_scale</i>	Scaling factor.

Value Returned

<i>l_scaledBBox</i>	Result.
---------------------	---------

Example

This example starts with a bounding box that is 100 units high by 200 units wide and gets two bounding boxes with the same center point, one whose sides are 50 units by 100 units and the other whose sides are 200 units by 400 units.

```
let( (myBBox list( 0:0 200:100 )) smallerBBox biggerBBox)  
    smallerBBox = hiScaleBox(myBBox 2)  
    ; result is ((50 25) (150 75)), which is 1/2 the dimensions  
    biggerBBox = hiScaleBox(myBBox .5)
```


Cadence User Interface SKILL Functions Reference

Command Interpreter Window

```
    ; result is ((-100.0 -50.0) (300.0 150.0)), which is  
    ; twice the dimensions  
)
```

Note that the return values are floating-point numbers in this case, even though the original bounding box was comprised of integers.

Reference

[hiBoxCenter](#)

hiSetBeepVolume

```
hiSetBeepVolume(  
    [x_loudness]  
)  
=> t
```

Description

Sets the loudness of the beep.

Arguments

x_loudness An integer representing the loudness of the beep. *x_loudness* ranges from -100 (silence) to 100 (full volume). The default is zero. Setting *x_loudness* to zero causes the beep to be at the system default loudness. The absolute value of a negative number is the percentage from the system default loudness toward silence, and a positive number is the percentage from the system default loudness toward full volume. The amount of loudness control is system dependent. For example, Sun systems have the same volume for values -99 through 100, but HP systems have a wide range of volume.

If you set *x_loudness* to an integer less than -100, it will be set to -100. If you set it to an integer more than 100, it will be set to 100.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiSetMultiClickTime

```
hiSetMultiClickTime(  
    x_milliseconds  
)  
=> t
```

Description

Sets the number of milliseconds that must elapse before a mouse click is recognized as a separate single click rather than the second click of a double click.

The valid range is 50–1000 (1/20th of a second to one second). Values below 50 are set to 50 and a warning is output. Values above 1000 are set to 1000 and a warning is output.

Arguments

<i>x_milliseconds</i>	Number of milliseconds that should elapse before another mouse click is recognized as a separate single click. Valid values: 50–1000.
-----------------------	--

Value Returned

t	Returns t after the time is set.
---	----------------------------------

Reference

[hiGetMultiClickTime](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiSetUserPreferences

```
hiSetUserPreferences( )  
=> t / nil
```

Description

Displays the User Preferences form with the latest values.

Before displaying the form, `hiSetUserPreferences` checks for changes to the form values that might have been made through SKILL or environment variables and synchronizes the form with those values.

Arguments

None.

Value Returned

`t` Returns `t` after the values are set.

`nil` Returns `nil` if the operation failed.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiSynchronize

```
hiSynchronize(  
    g_ignored  
)  
=> t
```

Description

Calls the XWindows C-language function `XSynchronize` to synchronize event queues. The argument is required and ignored.

Arguments

<i>g_ignored</i>	Any valid argument, such as <code>t</code> or <code>nil</code> .
------------------	--

Value Returned

<i>t</i>	Returns <code>t</code> after <code>XSynchronize</code> is called.
----------	---

Reference

[hiFlush](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiViewfile

```
hiViewfile( )  
=> t / nil
```

Description

Brings up a form in which you can specify a file to view and then calls the `view` function on the file.

Arguments

None.

Value Returned

`t` Returns `t` after the form is closed if you clicked `OK` on the form.

`nil` Returns `nil` after the form is closed if you clicked `Cancel` on the form.

Process Control Functions

This section describes functions related to the global abort flag. This flag indicates whether the user has pressed *Ctrl-C* to stop a Cadence software process, but it can also be set programatically. Functions are listed in alphabetical order.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiCheckAbort

```
hiCheckAbort( )  
=> t / nil
```

Description

Returns the value of the global abort flag.

This flag can be set to `t` by calling `hiSetAbort` and to `nil` by calling `hiResetAbort`. The flag normally is set to `t` when the SKILL interrupt flag is set to `t` (typically when a user presses *Ctrl-C* in an Design Framework II window). The global abort flag, once set, remains set to `t` unless specifically reset using the SKILL routine `hiResetAbort` or when cleared by Cadence applications.

Arguments

None.

Value Returned

`t` Returns `t` if the flag is set to `t`.

`nil` Returns `nil` if the flag is set to `nil`.

Reference

[hiGetUserAbort](#), [hiSetAbort](#), [hiResetAbort](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetUserAbort

```
hiGetUserAbort( )  
=> t / nil
```

Description

Returns `t` if the user has pressed *Ctrl-C* to abort a procedure, and `nil` otherwise.

This function also updates the “tumbling dice” cursor. In procedures that are compute-intensive, you should call `hiGetUserAbort` at least once a second to update the cursor and inform the user that processing has not stopped.

Normally, if the function returns `t`, you should abort processing and return control to the user.

Arguments

None.

Value Returned

<code>t</code>	Returns <code>t</code> if the user has pressed <i>Ctrl-C</i> .
<code>nil</code>	Returns <code>nil</code> if the user has not pressed <i>Ctrl-C</i> .

Reference

[hiCheckAbort](#), [hiSetAbort](#), [hiResetAbort](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiResetAbort

```
hiResetAbort( )  
=> t
```

Description

Sets the global abort flag to `nil`.

Arguments

None.

Value Returned

t The flag is set to `nil`.

Reference

[hiCheckAbort](#), [hiGetUserAbort](#), [hiSetAbort](#)

hiSetAbort

```
hiSetAbort( )  
=> t
```

Description

Sets the global abort flag to `t`.

Arguments

None.

Value Returned

`t` Returns `t` if the global abort flag is set to `t`.

Reference

[hiCheckAbort](#), [hiGetUserAbort](#), [hiResetAbort](#)

Log File Functions

This section describes functions related to the main session log file, `CDS.log`, and ancillary user-created log files. Functions are listed alphabetically.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiEndLog

```
hiEndLog(  
    )  
=> t / nil
```

Description

Stops writing the transaction log to the file previously specified by `hiStartLog`.

Use `hiStartLog` and `hiEndLog` to create short ancillary log files for specific operations. All CIW transactions continue to be logged in the main log file `CDS.log`.

Arguments

None.

Value Returned

<code>t</code>	Returns <code>t</code> if the transaction log is no longer being written to the file.
<code>nil</code>	Returns <code>nil</code> and issues an error message if writing to the specified file is not stopped.

Reference

[hiStartLog](#)

hiGetLogFileName

```
hiGetLogFileName( )  
=> t_logPath
```

Description

Returns the rooted file name of the Design Framework II session log file.

Arguments

None.

Value Returned

t_logPath Rooted file name of the log file.

Reference

[hiReplayFile](#), [hiIsInReplay](#), [hiSetFilter](#), [hiSetFilterOptions](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiIsInReplay

```
hiIsInReplay( )  
=> t/nil
```

Description

Indicates whether your Design Framework II (DFII) session is in replay mode, that is, if commands are being executed from a replay file, which is typically a log file. Your session is in replay mode if you are running a DFII workbench with the `-replay logfile` or `-test` option or if you are replaying a file with the `hiReplayFile` SKILL function.

The `hiIsInReplay` function returns `t` while commands are being executed from the replay file. When the end of the file is reached and normal, interactive use of the CIW is possible, the function returns `nil`.

Arguments

None.

Value Returned

<code>t</code>	Your DFII session is in replay mode.
<code>nil</code>	Your DFII session is not in replay mode.

Reference

[hiReplayFile](#)

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiReplayFile

```
hiReplayFile(  
    s_fileName  
)  
=> t / nil
```

Description

Replays a log file. Use this function only for testing or debugging. In general, use the functions `load` or `loadi` instead of `hiReplayFile`. `hiReplayFile` is intended to be run only from a Command Interpreter Window.

Only the input lines (prefixed by `\i`), the accelerated input lines (prefixed by `\a`), and lines that have no backslash prefix are filtered from the replay file and sent to SKILL.

If `hiReplayFile` is used in a SKILL file, all the commands in the SKILL file, including the ones that follow `hiReplayFile`, will be executed before the commands in the replay file. If the SKILL file is embedded in another SKILL file, all the commands in that file will also be executed before the commands in the replay file.

Because `hiReplayFile` replays a log file, to read a SKILL file (and execute functions in it), use the functions `load` or `loadi` instead. These functions are described in the [SKILL Language Reference](#).

Note: `hiReplayFile` will not execute from a replay file (a file being replayed by the `-replay` command line argument or by using `hiReplayFile`).

Arguments

s_fileName The file to be read.

Note: Do not specify a file that has the `hiReplayFile` function in it.

Value Returned

t Returns *t* if the file is read.

nil Returns *nil* and issues an error message if the file is not read.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Reference

hiIsInReplay, hiGetLogFileName

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiSetFilter

```
hiSetFilter(  
    [s_filter]  
)  
=> t / nil
```

Description

Sets the output message filter for the CIW with the option specified by *s_filter*. If no option is specified, a form appears allowing you to specify the display options you want.

Program output, typed output, and error and warning messages are displayed by default.

Arguments

s_filter Filter setting for the CIW. The following options are available:

Filter	Explanation
default	Only program output, user typed output, and error and warning messages are displayed.
error	Only error and warning messages are displayed.
verbose	Only menu command input, program output, menu command output, and error and warning messages are displayed.
all	Menu command input, prompt output, program output, menu command output, and error and warning messages (everything except user typed output) are displayed.

Value Returned

t Returns *t* if the output message filter is set.

nil Returns *nil* and issues an error message if the output message filter is not set.

hiSetFilterOptions

```
hiSetFilterOptions(  
    g_inputMenuCommands  
    g_inputPrompts  
    g_outputProgramResults  
    g_outputMenuCommands  
    g_outputUser  
    g_messageErrors  
    g_messageWarnings  
)  
=> t / nil
```

Description

Sets the output message filter for the CIW with the specified options.

Corresponds to the *Log Filter* command on the Set Options menu. Program output, typed output, error and warning messages are displayed by default.

Arguments

<i>g_inputMenuCommands</i>	t or nil; default: nil
<i>g_inputPrompts</i>	t or nil; default: nil
<i>g_outputProgramResults</i>	t or nil; default: nil
<i>g_outputMenuCommands</i>	t or nil; default: nil
<i>g_outputUser</i>	t or nil; default: nil
<i>g_messageErrors</i>	t or nil; default: nil
<i>g_messageWarnings</i>	t or nil; default: nil

Note: The defaults listed are the values in effect at start-up. None of the arguments are optional.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Value Returned

<code>t</code>	Returns <code>t</code> if the output message filter is set.
<code>nil</code>	Returns <code>nil</code> and issues an error message if the output message filter is not set.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiStartLog

```
hiStartLog(  
    t_filename  
)  
=> t / nil
```

Description

Writes a transaction log to a file.

Use `hiStartLog` and `hiEndLog` to create short ancillary log files for specific operations. All CIW transactions continue to be logged in the main log file `CDS.log`.

Arguments

<i>t_filename</i>	Name of file to which the transaction log is to be written.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the transaction log is written to the specified file.
----------	---

<i>nil</i>	Returns <i>nil</i> and issues an error message if the transaction log is not written to the specified file.
------------	---

Reference

[hiEndLog](#)

Resource Management Functions

This section describes functions that return resource values from various databases and resource files. Functions are listed alphabetically.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetBBoxResource

```
hiGetBBoxResource(  
    t_resourceName  
    [t_resourceClass]  
)  
=> l_bBoxSpec
```

Description

Returns the bounding box specification defined in the `.Xdefaults` file for the specified resource.

Refer to the [hiGetGeometryResource](#) function.

Arguments

<i>t_resourceName</i>	Name of the resource to look up in the resource database.
<i>t_resourceClass</i>	Resource class to further limit the number of potential matches in the resource database. (See X Window System documentation for a further discussion on X resource classes. For general use, a <i>resourceName</i> is usually sufficient.)

Value Returned

<i>l_bboxSpec</i>	Returns a <i>//ur</i> bounding box (list of two points) specification. The proper specification of a geometry resource consists of
-------------------	--

`<width>x<height>{+-}xOffset{+-}yOffset,`

where *x y* is relative to the upper left corner of the screen. A conversion is performed to convert the geometry resource to the *//ur* notation. Negative values are not returned. If a value resulting from this conversion is negative, 0 is substituted for that value.

hiGetGeometryResource

```
hiGetGeometryResource(  
    t_resourceName  
    [t_resourceClass]  
)  
=> l_resource
```

Description

Gets geometry resources from the X resource database.

You must preface your resource with *Opus*. The syntax for describing a standard geometry string is documented in the *Xlib Programming Manual*, and is of the form:

```
<width>x<height>{+-}<xoffset>{+-}<yoffset>
```

where items enclosed in <> are integers and items enclosed in { } are a set from which one item is allowed.

Note: The syntax accepts only positive "+" values. Also, the x and y values are relative to the upper **left** corner of the screen.

Arguments

<i>t_resourceName</i>	Name of the resource to look up in the resource database.
<i>t_resourceClass</i>	Resource class to further limit the number of potential matches in the resource database. (See X documentation for a further discussion on X resource classes. For general use, a <i>resourceName</i> is usually sufficient.)

Value Returned

<i>l_resource</i>	Returns the list(<i>x y width height</i>) value of the specified resource. Any component of the geometry specification not specified is returned as a <i>nil</i> list value. If a negative x, y, width, or height is specified, 0 is returned for that component.
<i>nil</i>	Returns <i>nil</i> if the specified resource does not exist.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Example

The following examples are for a `.Xdefaults` file specifying the following resources:

```
Opus.geo1: 600x400+100+400
Opus.geo2: +1+2
Opus.geo3: 600+99
Opus.geo4: 400+1-20
Opus.geo5: 400x600+2
hiGetGeometryResource("geo1")
=> list( 100 400 600 400 )
hiGetGeometryResource("geo2")
=> list( 1 2 nil nil )
hiGetGeometryResource("geo3")
=> list( 99 nil 600 nil )
hiGetGeometryResource("geo4")
=> list( 1 0 400 nil )
hiGetGeometryResource("geo5")
=> list( 2 nil 400 600 )
hiGetGeometryResource("notThere")
=> nil
```

hiGetStringResource

```
hiGetStringResource(  
    t_resourceName  
    [t_resourceClass]  
)  
=> g_resource / nil
```

Description

Gets resources from the resource database of the X Window System.

Arguments

<i>t_resourceName</i>	Name of the resource to look up in the resource database.
<i>t_resourceClass</i>	Resource class to further limit the number of potential matches in the resource database. (See X documentation for a further discussion on X resource classes. For general use, a <i>resourceName</i> is usually sufficient.)

Value Returned

<i>g_resource</i>	Returns the value of the specified resource.
<i>nil</i>	Returns <i>nil</i> if the resource does not exist.

Example

The following examples are for an `.Xdefaults` file specifying the following resources:

```
Opus.someString: hello  
Opus.someInt: 1  
hiGetStringResource("someString")  
=> "hello"  
hiGetStringResource("someInt")  
=> "1"  
hiGetStringResource("notThere")  
=> nil
```

Text Attribute Functions

This section describes functions for learning and setting font size and other text attributes. Functions are listed alphabetically.

hiGetFont

```
hiGetFont(  
    t_fontType  
)  
=> t_font / nil
```

Description

Returns the name of the font associated with the specified type font.

Arguments

<i>t_fontType</i>	One of the following strings: <code>text</code> , <code>error</code> , <code>label</code> , and <code>all</code> . (See hiSetFont for a description of each font type.)
-------------------	--

Value Returned

<i>t_font</i>	Returns the font associated with the specified <i>t_fontType</i> . If <i>t_fontType</i> is set to <i>all</i> , a list containing the test, label, and error fonts is returned (in that order).
<code>nil</code>	Returns <code>nil</code> and issues an error message if the specified fonts are not returned.

Example

```
hiGetFont("text")  
=> "9x15"
```

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiGetTextWidth

```
hiGetTextWidth(  
    t_fontName  
    t_text  
)  
=> x_width / -1
```

Description

Returns the width in pixels of a text string in a specified font.

This value is useful when creating two dimensional forms.

Arguments

<i>t_fontName</i>	Name of the font you want to use. (Various font names can be retrieved by typing the command <code>xlsfonts</code> in your <i>xterm</i> window.)
<i>t_text</i>	Any user-defined string.

Value Returned

<i>x_width</i>	Returns the width, in pixels, of the string when it is displayed in the font <i>t_fontName</i> .
-1	Returns -1 if <i>t_fontName</i> is not a valid font.

Example

```
hiGetTextWidth("9x15" "Cadence")  
=> 63
```

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiQueryFont

```
hiQueryFont(  
    t_fontName  
)  
=> l_fontAttributes
```

Description

Returns the values of the attributes of *t_fontName*.

Arguments

<i>t_fontName</i>	Name of the font you want to query. (Various font names can be retrieved by typing the command <code>xlsfonts</code> in your <i>xterm</i> window.)
-------------------	--

Value Returned

<i>l_fontAttributes</i>	Returns the font attributes in a disembodied property list. The attributes returned are:
-------------------------	--

fontName	String representing the font name
fontAscent	Integer representing the number of pixels above the font baseline
fontDescent	Integer representing the number of pixels below the font baseline
fontHeight	Integer representing the maximum height of the specified font (<i>fontAscent</i> + <i>fontDescent</i>)

Example

```
hiQueryFont("9x15")  
=> (nil fontName "9x15" fontAscent 12 fontDescent 3  
    fontHeight 15)
```

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiSetFont

```
hiSetFont(  
    [t_fontType]  
    [t_fontName]  
)  
=> t / nil
```

Description

Sets the font type to use the specified font *t_fontName*.

Call this function at start-up time or in your `.cdsinit` file. Calling the function in the middle of a session can produce unpredictable results.

Since most graphical objects are not designed to be resolution independent, changing the font for a session may not produce what you expect. Choosing fonts which are substantially larger than the default fonts can cause overlapping and/or clipping of graphical objects.

If neither argument is passed, *t_fontName* is not passed, or either argument is set to an empty string (""), the Set Font form is displayed and allows any of the valid fonts to be set.

Arguments

t_fontType

Valid font types are:

<code>text</code>	Font used in CIW and Encapsulation output windows, viewfiles, and the input area of text fields.
<code>error</code>	Font used for error messages. (This font type is currently unused.)
<code>label</code>	Font used in banner labels, menus, and forms.
<code>help</code>	Font used for text displayed in help dialog boxes.
<code>all</code>	Applies only when the font name is not supplied, or if it is passed as "".

When fontType `text` is set, the text displayed in the CIW and Encapsulation output windows is immediately redisplayed in the new font. Existing viewfile windows are not affected. Any viewfile windows, created after this routine is called, display the new font. The output of any text fields is not redisplayed immediately, but

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

only when you type a character in the text field. The text is then redisplayed with the new font.

When `fontType label` is set, any new menus or banner labels are generated with the new font. Existing menus and banner labels are not affected. All forms are displayed in the new font after being reexposed.

t_fontName Name of the font you want to set.

Value Returned

t Returns *t* if the fonts are set.

nil Returns *nil* and issues an error message if the font types are not valid.

Example

```
hiSetFont("text" "9x15")  
=> t
```

Sets the `text` font type to 9x15.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

hiTextWidth

```
hiTextWidth(  
    t_fontName  
    t_string  
)  
=> x_width / nil
```

Description

Determines the width, in pixels, of a text string if it were displayed in the font you specify. Returns `nil` if the font is not valid.

Argument

<i>t_fontName</i>	Name of the font you want to use.
<i>t_string</i>	Text string whose width you want to check in a particular font.

Value Returned

<i>x_width</i>	Returns the width, in pixels, of the string when it is displayed in the font <i>t_fontName</i> .
<i>nil</i>	Returns <code>nil</code> if <i>t_fontName</i> is not a valid font.

Customizing the Environment

The customization functions described in this section provide applications and users a generic means of registering, changing, retrieving, and saving environment variables.

Each function interacts with an environment database. The database consists of registered environment variables. To use an environment database, an application first registers variables with valid values. The database can be used to perform error checking, merging of environment data, and other bookkeeping functions.

An application may use the environment database for storing the state of its tool. This allows the data to be stored in a central repository instead of the UI components of each tool. It also provides a clean separation between the application's data and the components that are used to reflect the data.

.cdsenv for Applications

If applications use the environment variable database, they will create a tool specific file named `.cdsenv`, which defines each environment variable. This file must be located in the `your_install_dir/tools/dfII/etc/tools/aTool` directory, where `aTool` is the name of the application. Each line of the registration file has the following format:

```
tool[.partition] name dataType value private_flag {choices | minmax}  
tolerance "comment"
```

where

tool is the name of the tool; [*.partition*] is a way of grouping related variables under any partition name. For example, `cdsLibManager.copy`.

name is the name of the variable.

dataType is the data type of the variable.

value is the value of the variable.

private_flag is `t` or `nil` and specifies whether the variable will be saved into an output file when there is a request for storing the environment variable.

choices is the list of choices for a cyclic or toggle variable. *choices* must be specified for cyclic and toggle variables.

minmax is the minimum and maximum value for int and float variables. *minmax* is optional.

tolerance is the minimum difference between the current value and the default or registered value of a float variable that would indicate that the value has changed. Use

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

tolerance only for float variables—if you do not specify *tolerance* for a float variable, the default tolerance value is .000001.

"*comment*" is an optional comment.

For example, the format for a string variable is:

```
myTool[.partition] myStringVar string "some string" nil "string variable example"
```

If the variable is a cyclic or toggle variable, *choices* should follow *private_flag*. For example:

```
tools[.partition] myCyclicVar cyclic "a" nil ("a" "b" "c")
tools[.partition] myToggleVar toggle (t nil nil) nil ("ch1" "ch2" "ch3")
```

If the variable is an int or float variable, the *min* and *max* values, if any, should follow *private_flag*. For example:

```
tools[.partition] myIntVar int 55 nil 0 100 "int variable example"
```

If the variable is a float variable, the *tolerance* value, if any, should follow the *min* and *max* values. For example:

```
tools[.partition] myIntVar int 55 nil 0 100 .0001 "int variable example"
```

If you want to define *tolerance*, but not *min* and *max*, use *nil* for *min* and *max*.

Customizing Defaults

A customizable file is located in *your_install_dir/tools/dfII/samples/.cdsenv*. This file is a concatenated version of all tool registration files. It contains only non-private variables.

Customizing a copy of this *.cdsenv* file is a good starting point. When customizing *.cdsenv* on a group basis, store the customized copy in *your_install_dir/tools/dfII/local/.cdsenv*. For a personal customized file, make a copy of the *.cdsenv* file in *your_install_dir/tools/dfII/samples/* or *your_install_dir/tools/dfII/local/* and place it in your home directory.

The format of each line in the registration file is different from the lines of the customizable version. In the customizable version, there are blank lines, comment lines starting with a semicolon, and lines in the following format:

```
tool[.partition] name dataType value
```

For example, the text font setting for a tool called "ui" can be specified as follows:

```
; text font
ui textFont string "fixed"
```

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Note: Variables must first be registered so that the database can recognize any environment variables set in the `.cdsenv` file or referenced by the functions described in this section.

At first reference or specific request, the environment variables defined in the `.cdsenv` file are loaded from `.cdsenv` in the following order by default:

1. `your_install_dir/tools/dfII/etc/tools/Tool` (tool variables will be registered)
2. `your_install_dir/tools/dfII/local/` (load local settings)
3. home directory (load personal settings)

As new environment variables are loaded, the new variable values override any previous settings. It is possible to register trigger functions to be called when a tool's variables are loaded, set, or dumped. For example, after a tool's environment variables are loaded, any SKILL `loadProc` associated with that tool will be called so that, for example, its internal state can be updated with the new environment variable values.

The process of reading the `.cdsenv` file and initializing the variables is referred to as *faulting in*.

envCyclicStringToIndex

```
envCyclicStringToIndex(  
    t_tool[.Partition]  
    t_varName  
    t_cyclicString  
)  
=> x_itemIndex
```

Description

Returns the index of the string for the given cyclic variable. This order is determined by the order of choices specified in the tool's registration file. The index is 0-based. -1 will be returned for an invalid choice.

Arguments

<i>t_tool[.Partition]</i>	The name of the tool (application) and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<i>t_varName</i>	The name of the variable
<i>t_cyclicString</i>	The string whose index in the list of choices is desired.

Value Returned

<i>x_itemIndex</i>	0-based index of this string in the choice list. If the specified string cannot be found among the list of choices, or other error conditions occur, -1 will be returned.
--------------------	---

envGetAvailableTools

```
envGetAvailableTools(  
    )  
=> l_availableTools
```

Description

Allows users to know what tools are available to be loaded.

Arguments

None.

Value Returned

l_availableTools The list of tools that are available to be loaded.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envGetDefVal

```
envGetDefVal(  
    t_tool[.Partition]  
    t_varName  
    [s_varType]  
)  
=> g_data
```

Description

Retrieves the registered default value for a variable. The tool will be faulted in if it is not already faulted in.

Arguments

<i>t_tool[.Partition]</i>	The name of the tool (application) and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<i>t_varName</i>	The name of the variable.
<i>s_varType</i>	The type of the variable. Possible choices are 'int, 'float, 'string, 'boolean, 'cyclic or 'toggle.

Value Returned

g_data

Returns the value of the variable. The following types will be returned for the following data types:

Data Type	Returned Value Type
int	int
float	float
string	string
boolean	t/nil
cyclic	string
toggle	list of t/nil

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envGetLoadedTools

```
envGetLoadedTools(  
    )  
=> l_loadedTools
```

Description

Allows users to know what tools have currently been loaded.

Arguments

None.

Value Returned

l_loadedTools The list of loaded tools.

envGetModifiedTools

```
envGetModifiedTools(  
    )  
=> l_modToolList / nil
```

Description

Returns a list of all tools with modified variables.

Arguments

None.

Value Returned

<i>l_modToolList</i>	The list of tools that have modified variables.
<i>nil</i>	Returns <i>nil</i> if there are no tools with modified variables.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envGetVal

```
envGetVal(  
    t_tool[.Partition]  
    t_varName  
    [s_varType]  
)  
=> g_data
```

Description

Retrieves the value of an environment variable. The tool will be faulted in if it is not already faulted in.

Arguments

<i>t_tool[.Partition]</i>	The name of the tool and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<i>t_varName</i>	The name of the variable.
<i>s_varType</i>	The type of the variable. Possible choices are 'int, 'float, 'string, 'boolean, 'cyclic or 'toggle.

Value Returned

g_data Returns the value of the variable. The following types will be returned for the following data types:

Data Type	Returned Value Type
int	int
float	float
string	string
boolean	t/nil
cyclic	string
toggle	list of t/nil

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envGetVarType

```
envGetVarType(  
    t_tool[.Partition]  
    t_varName  
)  
=> s_varType
```

Description

Returns the type of the variable specified. This query will cause the tool to be loaded, if it has not already been loaded.

Arguments

<i>t_tool[.Partition]</i>	The name of the tool (application) and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<i>t_varName</i>	The name of the variable.

Value Returned

<i>s_varType</i>	The type of the variable. Possible choices are 'int, 'float, 'string, 'boolean, 'cyclic, 'toggle or 'unknown.
------------------	---

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envIsToolModified

```
envIsToolModified(  
    t_toolName  
)  
=> t / nil
```

Description

Checks if any of the variables for a specified tool have been modified.

Arguments

<i>t_toolName</i>	The name of the tool.
-------------------	-----------------------

Value Returned

<i>t</i>	Returns <i>t</i> if any of the variables for the specified tool have been modified.
<i>nil</i>	Returns <i>nil</i> if none of the variables for the specified tool have been modified.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envIsVal

```
envIsVal(  
    t_tool[.Partition]  
    t_varName  
    [g_faultIn]  
)  
=> t / nil
```

Description

Indicates whether a particular variable has been loaded and whether it exists for the given *tool[.partition]*.

Arguments

<i>t_tool[.Partition]</i>	The name of the tool and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<i>t_varName</i>	The name of the variable.
<i>g_faultIn</i>	Determines whether to fault the tool in if the variable is not loaded. Faulting in is the process of reading the <i>.cdsenv</i> file and initializing the variables.

Value Returned

<i>t</i>	Returns <i>t</i> if the variable is currently loaded. If not, the <i>faultIn</i> flag will determine whether to fault that tool in and then the query for the existence of the specified variable will take place.
<i>nil</i>	Returns <i>nil</i> if the variable is not loaded.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envLoadFile

```
envLoadFile(  
    t_fileName  
)  
=> t / nil
```

Description

Specifies that a set of environment variables is to be loaded from a specified file. This function is equivalent to calling `envLoadVals()` with the `?tool` argument set to `ALL`. All values loaded will be marked as modified.

Arguments

<i>t_fileName</i>	The name of the file from which a set of environment variables will be loaded.
-------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the file is added to the list of files to search.
<i>nil</i>	Returns <i>nil</i> if the file could not be added to the list of files to search.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envLoadVals

```
envLoadVals(  
    ?envFile t_fileName  
    [?tool t_toolName]  
)  
=> t / nil
```

Description

Specifies that a set of environment variables is to be loaded from a specified file.

This file name will be added to the list of files to search, which is set to *your_install_dir/local/.cdsenv* and *~/cdsenv* upon startup. When a new tool is loaded, the list of files will be searched for the specified tool's variables. The variables for each subsequent file on the list overwrite any previous file values.

Arguments

<i>t_fileName</i>	The name of the file from which a set of environment variables will be loaded.
<i>t_toolName</i>	The tool for which you want the environment variables to be loaded. The default is an empty string (""). If no tools are specified, all the variables associated with the tools that are already loaded will be loaded from the specified file. If "ALL" is specified as <i>toolName</i> , the variables from all tools listed in this file will be loaded (possibly for the first time). Only if "ALL" is not specified as the tool name, the file will be added to the list of files to load from when a new tool is invoked, following the <i>.cdsenv</i> files in <i>cdshier/etc/tool</i> directory, <i>cdshier/local</i> directory, and the home directory. For more information, see the <u>envLoadFile()</u> function.

Value Returned

<i>t</i>	Returns <i>t</i> if the file is added to the list of files to search.
<i>nil</i>	Returns <i>nil</i> if the file could not be added to the list of files to search.

envRegLoadDumpTrigger

```
envRegLoadDumpTrigger(  
    ?tool t_toolName  
    ?loadFunc s_loadFunc  
    ?dumpFunc s_dumpFunc  
    ?preDumpFunc s_preDumpFunc  
)  
=> t / nil
```

Description

Registers the trigger function to be called when a tool's environment variables are loaded or saved to a file. Both the load and dump trigger functions are called after the load or dump of the environment variables, respectively. The preDump trigger is called just before the dump occurs. The dump trigger can be used to update the *defValues* of forms when a saving of defaults is performed.

Arguments

<i>t_toolName</i>	The name of the tool whose environment variables you want to store. If no tool is specified, all environment variables will be stored.
<i>s_loadFunc</i>	The function to be called after the tool's environment variables are loaded.
<i>s_dumpFunc</i>	The function to be called after the tool's environment variables are saved to a file.
<i>s_preDumpFunc</i>	The function to be called before any values are stored for the tool.

Value Returned

t	The trigger functions were registered.
nil	There was an error.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envRegSetTrigger

```
envRegSetTrigger(  
    t_tool  
    s_triggerFunc  
)  
=> t / nil
```

Description

Registers a set trigger function for a tool. This trigger will be called after an environment variable is set.

Arguments

<i>t_tool</i>	The name of the tool for which you want to register a set trigger function.
<i>s_triggerFunc</i>	The trigger function to be called after an environment variable is set. The trigger function is passed the variable that was set and its old and new values. The trigger function has the following format: <i>setTrigger(t_varName g_oldValue g_newValue)</i>

Value Returned

t	The set trigger function was registered.
nil	There was an error.

envSetToolCurrValToDefault

```
envSetToolCurrValToDefault(  
    t_tool[.Partition]  
)  
=> t / nil
```

Description

Sets the current value of all the environment variables to the default values.

Arguments

<code>t_tool[.Partition]</code>	The name of the tool and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
---------------------------------	---

Value Returned

<code>t</code>	The variables were reset.
<code>nil</code>	The variables could not be reset.

envSetToolDefaultToCurrVal

```
envSetToolDefaultToCurrVal(  
    t_tool[.Partition]  
)  
=> t / nil
```

Description

Sets the default value of all the environment variables to the current value of the variables.

Arguments

t_tool[.Partition] The name of the tool and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.

You can specify *t_tool[.partition]* in the following ways:

tool (resets all the variables at the top level of the tool and in all the partitions)

tool. (resets only the variables at the top level of the tool; does not reset the variables in the partitions)

tool.partition (resets only the variables in the specified partition)

Value Returned

t The variables were reset.

nil The variables could not be reset.

envSetVal

```
envSetVal(  
    t_tool[.Partition]  
    t_varName  
    s_varType  
    g_newVal  
)  
=> t / nil
```

Description

Sets an environment variable value in the setup database. The tool will be faulted in if it is not already faulted in—faulting in is the process of reading the `.cdsenv` file and initializing the variables. If a set trigger function is registered for this variable (see the `envRegSetTrigger` function), it will be called after the value is set.

Note: You cannot set an environment variable that has not been registered.

Arguments

<code>t_tool[.Partition]</code>	The name of the tool and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<code>t_varName</code>	The name of the environment variable.
<code>s_varType</code>	The type of the variable. Valid choices are 'int', 'float', 'string', 'boolean', 'cyclic' or 'toggle.
<code>g_newVal</code>	The new value for the environment variable.

Value Returned

<code>t</code>	The new value was set.
<code>nil</code>	There was an error and the new value could not be set.

envSetVarCurrValToDefault

```
envSetVarCurrValToDefault(  
    t_tool[.Partition]  
    t_varName  
)  
=> t / nil
```

Description

Sets the current value of the environment variable you specify to the default value.

Arguments

<i>t_tool[.Partition]</i>	The name of the tool and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<i>t_varName</i>	The name of the environment variable.

Value Returned

<i>t</i>	The variables were reset.
<i>nil</i>	The variables could not be reset.

envSetVarDefaultToCurrVal

```
envSetVarDefaultToCurrVal(  
    t_tool[.Partition]  
    t_varName  
)  
=> t / nil
```

Description

Sets the default value of the environment variable you specify to the current value of the variable.

Arguments

<code>t_tool[.Partition]</code>	The name of the tool and the partition (if any). A partition is a way of grouping related sets of variables associated with a tool.
<code>t_varName</code>	The name of the environment variable.

Value Returned

<code>t</code>	The variables were reset.
<code>nil</code>	There was an error.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

envStoreEnv

```
envStoreEnv(  
    [?envFile t_fileName]  
    [?tool t_tool[.partition]]  
    [?toolStatus t_toolStatus]  
    [?varStatus s_varStatus]  
    [?fileStatus t_fileStatus]  
)  
=> t / nil
```

Description

Stores environment variables associated with tools to a specific file. You can choose to save variables for a specific tool, all loaded tools, or all possible tools. You can also specify whether you want to save all the variables for a tool, the variables that have been modified, or only the variables that have been modified and that are different from the default value.

This function is the programmatic equivalent of saving variables with the Save Defaults form.

To open the Save Defaults form,

- In the CIW, choose *Options – Save Defaults*.

The 'Save Defaults' dialog box is shown with a title bar and standard window controls. It contains several sections for configuring the save operation:

- Buttons:** OK, Cancel, Defaults, Apply, and Help.
- Tools To Save:** Three radio buttons: 'All possible tools' (unselected), 'All loaded tools' (selected), and 'Loaded tools by name' (unselected). Below the radio buttons is a text field labeled '(fill in Tool Names)'.
- Variables To Save:** Three radio buttons: 'Modified variables only' (selected), 'Modified and different from Default' (unselected), and 'All tool variables' (unselected).
- Available Tools:** A list box containing: adle, amsDirect, asimenv, ats, auCdl, auCore, and auLvs.
- Currently Loaded:** A list box containing: auCore, ddserv, graphic, rod, schematic, and ui.
- Tool Names:** A text field containing 'ALL'.
- Save To File:** A text field containing '~/.cdserv'.
- File Status:** Three radio buttons: 'Overwrite' (unselected), 'Merge values' (selected), and 'Retain values' (unselected).

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

Arguments

t_fileName The name of the file in which you want to store the environment variables. If a file name is not specified, the variables will be saved in `~/cdsenv`.

t_tool[.partition] The name of the tool and the partition, if any. Use this argument only if you have set *t_toolStatus* to "Loaded tools by name".

A partition is a way of grouping related sets of variables associated with a tool. You can specify the value of this argument in the following ways:

tool (saves all the variables at the top level of the tool and in all the partitions)

tool. (saves only the variables at the top level of the tool; does not save the variables in the partitions)

tool.partition (saves only the variables in the specified partition)

If you specify more than one tool, separate the names with a space.

The default value of this argument is "ALL".

t_toolStatus The tools you want to save. Specify one of the following strings:

"All possible tools"

"All loaded tools"

"Loaded tools by name"

If you specify "Loaded tools by name," specify the names of the tools in the *t_tool[.partition]* argument.

If you specify "All possible tools," the value of the *s_varStatus* argument is ignored. All the variables are saved.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

The default value is "All loaded tools".

s_varStatus

The variables you want to save. Specify one of the following symbols:

'modified (saves only those variables that have been modified; a variable is considered to be modified if it is changed in any way, even if its value is reset to the original value)

'modifiedFromDefault (saves only those variables that have been modified and that are different from the default; float variables will be saved if the difference between the current value of the variable and the default value is greater than the tolerance specified in the .cdsenv file)

'all (saves all the variables)

Note: If you specified "All possible tools" as the value of the *t_toolStatus* argument, the *s_varStatus* argument is ignored. *s_varStatus* is assumed to be 'all, regardless of the value you specify.

t_fileStatus

The file status. Specify one of the following strings:

"Overwrite" (overwrites the original file)

"Merge values" (merges values that have been modified in the current session with the original file)

"Retain values" (retains values from the original file)

The default is "Merge values".

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

The following table shows the values that will be saved to an existing file based on the values of *s_varStatus* and *t_fileStatus*:

	Merge Values	Overwrite	Retain values
Modified	All values in the existing default settings file will be retained unless the variable has been modified in the current session. If the variable has been modified, its modified value will be saved. In addition, non-private values that are not included in the original file but that have been modified will be added to the file.	Only non-private values that have been modified in the current session will be saved. Any pre-existing contents of the file will be removed.	All values in the existing default settings file will be retained. Any additional values that are not included in the original file but that have been modified in the current session will be added to the file.
Modified From Default	All values in the existing default settings file will be retained unless the variable has been modified in the current session and is different from the default value. If the variable has been modified and its value is different from the default value, its new value will be saved. In addition, non-private values that are not included in the original file will be added to the file if they have been modified and are different from the defaults.	Only non-private values that have been modified in the current session and that are different from the default values will be saved. Any pre-existing contents of the file will be removed.	All values in the existing default settings file will be retained. In addition, any non-private values that are not included in the original file will be added to the file if they have been modified and are different from the defaults.

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

	Merge Values	Overwrite	Retain values
All	All values in the existing default settings file will be retained unless the variable has been modified in the current session. If the variable has been modified, its modified value is saved. In addition, all non-private values for the tool will be written to the file, whether or not they have been used in the current session.	All non-private values for the tool will be saved whether or not they are used in the current session. Any pre-existing contents of the file will be removed.	All values in the existing default settings file will be retained. In addition, all non-private values for the tool will be written to the file, whether or not they have been used in the current session.

Value Returned

<code>t</code>	The environment variables were saved in the file you specified.
<code>nil</code>	There was an error.

Example

The following example shows different environment variables that will be saved to the existing file (`myDefs`) based on the values of `t_toolStatus`, `t_tool[.partition]`, `s_varStatus`, and `t_fileStatus`. Assume that the list of all possible tools includes `testTool`, `anotherTool` and `tool3`.

Master variable registration file located in `etc/tools/testTool/.cdsenv`:

```
testTool    showPromptLines    boolean    t        nil
testTool    nestLimit           int        5        nil
testTool    showScrollBars      boolean    nil       nil
testTool    setWinTraversal      boolean    nil       nil
testTool    undoLevel           int        1        nil
```

Master variable registration file located in `etc/tools/anotherTool/.cdsenv`:

```
anotherTool var1    string    "hi"    nil
anotherTool var2    int      50      nil
anotherTool var3    cyclic   "v2"    nil ("v1" "v2" "v3")
```

Master variable registration file located in `etc/tools/tool3/.cdsenv`:

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

```
tool3.part1  check  boolean nil  nil
tool3.partb  level  int    20    nil
tool3        auto   boolean t    nil
```

Existing file `myDefs`:

```
testTool      showPromptLines  boolean nil
testTool      nestLimit         int 3
testTool      undoLevel         int 5
tool3         auto              boolean nil
tool3.partb    level            int 9
anotherTool    var3              cyclic  "v3"
```

Environment variable settings in the current session:

```
testTool  showPromptLines  boolean nil  (modified and different from default)
testTool  nestLimit        int 2        (modified and different from default)
testTool  showScrollBars   boolean nil
testTool  setWinTraversal   boolean t    (modified and different from default)
testTool  undoLevel        int 1
tool3     auto              boolean t    (modified and same as default)
```

The following is a list of environment variables that will be saved to file `myDefs` depending on the settings for `t_fileStatus`, `t_tool[.partition]`, `s_varStatus` and `t_toolStatus`.

For `?fileStatus` **"Overwrite"**, `?toolStatus` **"Loaded tools by name"**, `?tool` **"testTool tool3"**, and `?varStatus` **'modified'**:

```
testTool  showPromptLines  boolean  nil
testTool  nestLimit        int      2
testTool  setWinTraversal   boolean  t
tool3     auto              boolean  t
```

For `?fileStatus` **"Merge values"**, `?toolStatus` **"Loaded tools by name"**, `?tool` **"testTool tool3"**, and `?varStatus` **'modified'**:

```
testTool      showPromptLines  boolean  nil
testTool      nestLimit        int      2
testTool      setWinTraversal   boolean  t
testTool      undoLevel        int      5
tool3         auto              boolean  t
tool3.partb    level            int      9
anotherTool    var3              cyclic   "v3"
```

For `?fileStatus` **"Retain values"**, `?toolStatus` **"Loaded tools by name"**, `?tool` **"testTool tool3"**, and `?varStatus` **'modified'**:

```
testTool      showPromptLines  boolean  nil
testTool      nestLimit        int      3
testTool      setWinTraversal   boolean  t
testTool      undoLevel        int      5
tool3         auto              boolean  nil
tool3.partb    level            int      9
anotherTool    var3              cyclic   "v3"
```

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

For ?fileStatus **"Overwrite"**, ?toolStatus **"Loaded tools by name"**, ?tool **"testTool tool3"**, and ?varStatus **'modifiedFromDefault'**:

testTool	showPromptLines	boolean	nil
testTool	nestLimit	int	2
testTool	setWinTraversal	boolean	t

For ?fileStatus **"Overwrite"** and ?toolStatus **"All possible tools"**:

testTool	showPromptLines	boolean	nil
testTool	nestLimit	int	2
testTool	showScrollBars	boolean	nil
testTool	setWinTraversal	boolean	t
testTool	undoLevel	int	1
anotherTool	var1	string	"hi"
anotherTool	var2	int	50
anotherTool	var3	cyclic	"v2"
tool3.part1	check	boolean	nil
tool3.partb	level	int	20
tool3	auto	boolean	t

For ?fileStatus **"Merge values"** and ?toolStatus **"All possible tools"**:

testTool	showPromptLines	boolean	nil
testTool	nestLimit	int	2
testTool	showScrollBars	boolean	nil
testTool	setWinTraversal	boolean	t
testTool	undoLevel	int	5
anotherTool	var1	string	"hi"
anotherTool	var2	int	50
anotherTool	var3	cyclic	"v2"
tool3.part1	check	boolean	nil
tool3.partb	level	int	9
tool3	auto	boolean	t

For ?fileStatus **"Retain values"** and ?toolStatus **"All possible tools"**:

testTool	showPromptLines	boolean	nil
testTool	nestLimit	int	3
testTool	showScrollBars	boolean	nil
testTool	setWinTraversal	boolean	t
testTool	undoLevel	int	5
anotherTool	var1	string	"hi"
anotherTool	var2	int	50
anotherTool	var3	cyclic	"v3"
tool3.part1	check	boolean	nil
tool3.partb	level	int	9
tool3	auto	boolean	nil

For ?fileStatus **"Overwrite"**, ?toolStatus **"Loaded tools by name"**, ?tool **"tool3"**, and ?varStatus **'all'**:

tool3	auto	boolean	t
tool3.part1	check	boolean	nil
tool3.partb	level	int	20

For ?fileStatus **"Overwrite"**, ?toolStatus **"Loaded tools by name"**, ?tool **"tool3."**, and ?varStatus **'all'**:

Cadence User Interface SKILL Functions Reference

Command Interpreter Window

tool3 auto boolean t

For ?fileStatus **"Overwrite"**, ?toolStatus **"Loaded tools by name"**, ?tool
"tool3.part1", and ?varStatus **'all'**:

tool3.part1 check boolean nil

Menus

The following topics are discussed in this chapter:

- [Overview](#) on page 120
- [Menu Functions](#) on page 122
- [Programming Samples](#) on page 163

Overview

There are three varieties of menus that you can create using SKILL:

- Popup menus
- Fixed menus
- Pulldown menus

Popup Menu

You can create three different kinds of popup menus using `hiCreateSimpleMenu`, `hiCreateMenu`, and `hiCreate2DMenu`. The behavior and appearance of menus created with these three SKILL routines is described below:

`hiCreateSimpleMenu`

Creates a menu with text entries only. You can generate this menu and its entries at once, without the need to create menu items up front.

`hiCreateMenu`

Creates a menu with text, icon, or slider menu items. Menu items are laid out in a column, top to bottom, in a specified order. Items can also be disabled (grayed-out) so that they are unselectable.

`hiCreate2DMenu`

Creates a menu with text or icon *active* menu items, or *inactive* labels. This menu type lets you specify the exact placement of each menu item through an attribute list supplied with each item at creation.

Fixed Menus

Fixed menus can be generated using `hiCreateVerticalFixedMenu` and `hiCreateHorizontalFixedMenu`.

Both create a menu that is laid out in column/row order. Fixed menus typically occupy the entire top, bottom, or side of the screen. Vertical fixed menus may also appear within graphics windows. You can specify the number of rows and columns at creation.

Pulldown Menu

For a pulldown menu, you use `hiCreatePulldownMenu`, which creates a menu that is described in the same manner as `hiCreateMenu`. This menu differs from popup menus because you have to insert it into the banner of any created window (refer to Window Management Functions), then pull it down from the banner to display it.

All menus require a *menuHandle*, which is a SKILL handle to the menu. With the exception of simple menus, all menu creation routines require a list of menu items. You must have created these previously using `hiCreateMenuItem` or `hiCreateSliderMenuItem`. If you select a non-slider menu item, the system executes a command. If you select a slider menu item, a submenu is displayed. Whenever a menu item is selected from *non-fixed* menus, whether or not from a submenu, the menu is brought down. Fixed menus, on the other hand, remain fixed on the screen until you select *Done*.

Help Menus

You create Help menus the same way you create other menus. When you create a Help menu, use "Help" as the menu title. This enables the Help menu to be placed at the right end of the menu bar.

Menu Functions

The following functions are listed in alphabetical order.

Cadence User Interface SKILL Functions Reference

Menus

hiAddMenuItem

```
hiAddMenuItem(  
    r_hiMenu  
    r_menuItem  
)  
=> t / nil
```

Description

Adds the *menuItem* to the end of all instances of the specified *hiMenu*.

Arguments

<i>r_hiMenu</i>	Menu created with <u>hiCreateMenu</u> or <u>hiCreatePulldownMenu</u> or <u>hiCreateSeparatorMenuItem</u> .
<i>r_menuItem</i>	Menu item description returned from <u>hiCreateMenuItem</u> or <u>hiCreateSliderMenuItem</u> .

Value Returned

t	Returns t if successful.
nil	Returns nil and an error message if the menu item is not added.

hiCreate2DMenu

```
hiCreate2DMenu(  
    s_menuHandle  
    t_menuName  
    l_menuItems  
)  
=> r_hiMenu
```

Description

Returns an internal data structure describing a two dimensional menu object. Once this menu is created, you can display it with [_hiDisplayMenu](#).

Arguments

<i>s_menuHandle</i>	A unique global SKILL symbol used to reference this menu.
<i>t_menuName</i>	Name of the menu that displays in the menu banner.
<i>l_menuItems</i>	List of menu items, each item being a list containing the 2D representation of a form button or label field. Each item list contains three elements: <ul style="list-style-type: none">❑ A button or label <i>form</i> field created by hiCreateButton or hiCreateLabel❑ The point where the upper-left corner of this field should be placed❑ A list containing the width and height of the field

See the section on [2D forms](#) for further explanation.

Value Returned

<i>r_hiMenu</i>	Returns the SKILL structure representing the menu.
-----------------	--

hiCreateHorizontalFixedMenu

```
hiCreateHorizontalFixedMenu(  
    s_menuHandle  
    l_menuItems  
    x_rows  
    x_cols  
)  
=> r_hiMenu
```

Description

Returns an internal data structure that describes a horizontal fixed menu object. The fixed menu is created as a grid of *row* x *column* dimensions, and contains menu item objects created by [hiCreateMenuItem](#). An additional *Done* item is added as the last item. Selecting this *Done* menu item will remove the fixed menu from the screen.

The width and height of the horizontal fixed menu is determined by the number of rows and columns in the menu. Once this menu is created, you can display it with [hiDisplayFixedMenu](#).

Horizontal fixed menus appear across the entire top or bottom of the root window (screen), rather than an application window. These fixed menus are shared among all Cadence application windows contained within the root window. Menu items you want to display within the fixed menu are inserted in row-major order. Keep this in mind when passing arguments to avoid graphically undesirable results.

Arguments

<i>s_menuHandle</i>	A unique global SKILL symbol used to reference this menu.
<i>l_menuItems</i>	List of menu items created by hiCreateMenuItem .
<i>x_rows</i>	Number of rows in the fixed menu.
<i>x_cols</i>	Number of columns in the fixed menu.

Value Returned

<i>r_hiMenu</i>	Returns the SKILL structure representing the menu. The <i>menuHandle</i> can be referenced by <i>menu->hiMenuSym</i> .
-----------------	---

Cadence User Interface SKILL Functions Reference

Menus

Example 1

Before using this example, see the [hiCreateMenuItem](#) example to find out how to create the menu items used in this example.

```
hiCreateHorizontalFixedMenu(  
    'trExampleHorizontalFixedMenu  
    list( trMenuItemOne trMenuItemTwo )  
    1  
    2  
)
```

For an example of displaying this menu, see [hiDisplayFixedMenu](#).

Example 2

If you need to create a series of menu items for the fixed menu, it is more efficient to create a symbol rather than creating each menu item individually.

```
Procedure( trCreateMenuItem( theMenuSymbol )  
    set(  
        theMenuSymbol  
        hiCreateMenuItem(  
            ?name theMenuSymbol  
            ?itemText get_pname( theMenuSymbol )  
            ?callback sprintf( nil "println( '%L )" theMenuSymbol )  
        )  
    ) ; set  
) ; procedure  
  
trCreateMenuItem( 'item1 )  
trCreateMenuItem( 'item2 )  
trCreateMenuItem( 'item3 )  
trCreateMenuItem( 'item4 )  
trCreateMenuItem( 'item5 )  
trCreateMenuItem( 'item6 )  
  
hiCreateHorizontalFixedMenu(  
    'trExampleHorizontalFixedMenu  
    list( item1 item2 item3 item4 item5 item6 )  
    1      ;; number of rows  
    6      ;; number of columns  
)  
  
hiDisplayFixedMenu(  
    trExampleHorizontalFixedMenu  
    "top"  
)  
  
hiCreateVerticalFixedMenu(  
    'trExampleVerticalFixedMenu  
    list( item1 item2 item3 item4 item5 item6 )  
    6      ;; number of rows  
    1      ;; number of columns  
)  
  
hiDisplayFixedMenu(  
    trExampleVerticalFixedMenu
```

Cadence User Interface SKILL Functions Reference

Menus

```
"left"  
)
```

Cadence User Interface SKILL Functions Reference

Menus

hiCreateMenu

```
hiCreateMenu(  
    s_menuHandle  
    t_menuTitle  
    l_menuItems  
)  
=> r_hiMenu
```

Description

Returns the SKILL representation of a popup menu. Once this menu is created, you can display it with [hiDisplayMenu](#) or associate it with a window using [hiSetWindowMenu](#).

Arguments

<i>s_menuHandle</i>	A unique global SKILL symbol used to reference this menu.
<i>t_menuTitle</i>	Title that displays in the menu banner.
<i>l_menuItems</i>	List of menu items, each created by hiCreateMenuItem or hiCreateSliderMenuItem or hiCreateSeparatorMenuItem .

Value Returned

<i>r_hiMenu</i>	Returns the SKILL structure representing the menu. The <i>menuHandle</i> can be referenced by <code>menu->hiMenuSym</code> .
-----------------	---

hiCreateMenuItem

```
hiCreateMenuItem(  
    ?name s_itemHandle  
    ?itemText t_menuItemText  
    [?itemIcon l_menuIcon]  
    [?callback t_itemCallback]  
    [?disable g_disabled]  
)  
=> r_hiMenuItem
```

Description

Creates a menu item that is used in the argument list to create any menu (except two-dimensional menus and simple menus).

This non-slider menu entry does not bring up any submenu objects. It can contain text (*t_menuItemText*) or icon (*l_menuIcon*) information, and it can have a user-defined SKILL callback procedure associated with its selection (*t_itemCallback*).

You can use menu items within pop-up or pull-down menus or as elements of a fixed menu. You can also use the same menu item within different menus.

If you create a menu with simple menu entries only (no icons or sliders), it is more efficient to use [hiCreateSimpleMenu](#).

In window fixed menus, if the *showFixedMenuLabels* property is set to *t* (`hiGetCIWindow()` -> `showFixedMenuLabels=t`), the menu item text (*t_menuItemText*) will be displayed below the menu icon (*l_menuIcon*) when the user moves the pointer into the icon. Displaying the text helps the user understand the meaning of the icon. This property can also be controlled from the User Preferences form of the CIW Utilities menu.

Note: Window fixed menus can only be used in Cadence “power windows,” such as graphics, browser, form, text, and encapsulation windows. (See [hiAddFixedMenu](#).)

Arguments

<i>s_itemHandle</i>	SKILL handle to the menu item.
<i>t_menuItemText</i>	String representing a text menu item.
<i>l_menuIcon</i>	List representing an icon menu item. You can create an icon list with hiStringToIcon or geCellViewToDlist and dIDlistToIcon . If you use <i>l_menuIcon</i> , any text in <i>t_menuItemText</i> is

Cadence User Interface SKILL Functions Reference

Menus

ignored (except in the case of window fixed menus when *hiGetCIWindow()->showFixedMenuLabels==t*).

t_itemCallback

String that represents a SKILL callback procedure to be executed when that menu item is selected.

g_disabled

If set to *t*, the menu item is disabled (grayed out and unselectable) when displayed; if not specified or *nil*, the menu item is enabled.

Value Returned

r_hiMenuItem

Returns the SKILL structure representing the menu item. The item handle and text menu item can be referenced by *menuItem->hiMenuItemSym* and *menuItem->hiItemText* respectively.

Example

```
trMenuItemOne = hiCreateMenuItem(  
  ?name 'trMenuItemOne  
  ?itemText "One"  
  ?callback "println( \"One\" )"  
)  
  
trMenuItemTwo = hiCreateMenuItem(  
  ?name 'trMenuItemTwo  
  ?itemText "Two"  
  ?callback "println( \"Two\" )"  
)
```

hiCreatePulldownMenu

```
hiCreatePulldownMenu(  
    s_menuHandle  
    g_menuTitle  
    l_menuItems  
)  
=> r_hiMenu
```

Description

Returns the SKILL description of a pulldown menu. It creates a menu that contains text, icon, or slider menu items. *s_menuHandle* is the SKILL symbol set to the created menu.

A pulldown menu is not explicitly displayed. You should place it on a window banner with [hiInsertBannerMenu](#), or you can insert a pulldown menu as a submenu using [hiCreateSliderMenuItem](#). You can share pulldown menus by placing the same menu in different windows.

Arguments

<i>s_menuHandle</i>	A unique global SKILL symbol used to reference this menu.
<i>g_menuTitle</i>	Title that displays in the menu banner. This can be either a text string or an icon list (see geCellViewToDlist and dldlistToIcon , or hiStringToIcon).
<i>l_menuItems</i>	List of menu items created by hiCreateMenuItem or hiCreateSliderMenuItem or hiCreateSeparatorMenuItem .

Value Returned

<i>r_hiMenu</i>	Returns the SKILL structure representing the menu. The <i>menuHandle</i> can be referenced by <i>menu->hiMenuSym</i> .
-----------------	---

Example

Before using this example, see the [hiCreateMenuItem](#) example to create the menu items used in this example.

```
hiCreatePulldownMenu(  
    'trPulldownMenu
```

Cadence User Interface SKILL Functions Reference

Menus

```
"Example Menu"  
list( trMenuItemOne trMenuItemTwo )  
)
```

For an example of displaying this pulldown menu, see [hiInsertBannerMenu](#).

hiCreateSimpleMenu

```
hiCreateSimpleMenu(  
    s_menuHandle  
    t_menuTitle  
    l_menuItems  
    l_menuItemCallbacks  
)  
=> r_hiMenu
```

Description

Returns the SKILL description of a *simple* menu, in that it is a single level menu containing only text menu items, and none of its menu entries invoke a submenu. Once this menu is created, you can display it with [hiDisplayMenu](#) or associate it with a window using [hiSetWindowMenu](#).

Arguments

<i>s_menuHandle</i>	A unique global SKILL symbol used to reference this menu.
<i>t_menuTitle</i>	Title that displays in the menu banner.
<i>l_menuItems</i>	List of strings that is contained within the simple menu.
<i>l_menuItemCallbacks</i>	List of string specifications of SKILL callback procedures that are called when the menu item is selected. There should be one callback string listed for every menu item. If only one callback string is specified, that routine is called back regardless of which menu item is selected. In this case, the text of the selected menu item will be appended to the callback string.

Value Returned

<i>r_hiMenu</i>	Returns the SKILL structure representing the menu. The <i>menuHandle</i> can be referenced by <i>menu->hiMenuSym</i> .
-----------------	---

Example

```
hiCreateSimpleMenu(  
    'trExampleMenu  
    "Example Menu"
```

Cadence User Interface SKILL Functions Reference

Menus

```
'( "One" "Two" )  
'( "println( \"One\" )" "println( \"Two\" )" ) )  
=> array[7]:27015368
```

For an example of displaying pop-up menus, see [hiDisplayMenu](#).

hiCreateSeparatorMenuItem

```
hiCreateSeparatorMenuItem(  
    ?name s_itemHandle  
)  
=> r_hiSeparatorMenuItem
```

Description

Creates a menu separator—a horizontal line that separates menu items. This menu item is used in the *l_menuItems* argument of the `hiCreateMenu` and `hiCreatePulldownMenu` functions. You can use the separator menu item in more than one menu or in multiple locations in the same menu.

Arguments

<i>s_itemHandle</i>	SKILL handle to the menu item.
---------------------	--------------------------------

Value Returned

<i>r_hiSeparatorMenuItem</i>	Returns the SKILL structure representing the separator menu item.
------------------------------	---

hiCreateSliderMenuItem

```
hiCreateSliderMenuItem(  
    ?name s_itemHandle  
    ?itemText t_menuItemText  
    ?subMenu r_pulldownMenu  
    [?itemIcon l_menuIcon]  
    [?disable g_disabled]  
)  
=> r_hiSliderItem
```

Description

Creates a slider menu item that is used in the argument list to create a menu (all menus except simple, fixed or 2D menus). Slider menu items bring up submenus when selected. They can contain text (*t_menuItemText*) or icon (*l_menuIcon*) information.

A slider menu entry is a composite object. As with all composite objects, you must create all its subobjects before making your slider menu entries. Call [hiCreateMenuItem](#) or [hiCreateSliderMenuItem](#) to create your pulldown menu items and then [hiCreatePulldownMenu](#) to generate your slider menu entry.

You can use slider menu items within a popup or pulldown menu. You can also share them by placing the same menu item within different menus.

Arguments

<i>s_itemHandle</i>	SKILL handle to the menu item.
<i>t_menuItemText</i>	String representing a text menu item.
<i>r_pulldownMenu</i>	Unlike hiCreateMenuItem , a user-defined callback procedure is not associated with slider menu items but with the menu items in <i>r_pulldownMenu</i> . <i>r_pulldownMenu</i> is a menu created with hiCreatePulldownMenu and displays when this menu item is selected.
<i>l_menuIcon</i>	List representing an icon menu item. An icon list can be created with hiStringToIcon or geCellViewToDlist and dldlistToIcon . If you use <i>menuIcon</i> , any text in <i>menuItemText</i> is ignored.

Cadence User Interface SKILL Functions Reference

Menus

g_disabled If set to `t`, this slider menu item will be disabled (grayed out and unselectable) when displayed; if not specified or `nil`, the menu item will be enabled.

Value Returned

r_hiSliderItem Returns the SKILL structure representing the menu item. The *itemHandle*, *menuItemText*, and *pulldownMenu* can be referenced by:

```
menuItem->hiMenuItemSym  
menuItem->hiItemText  
menuItem->hiSubMenu
```

Example

For an example of this function, see [“Creating a Sample Slider Menu” on page 168](#).

hiCreateVerticalFixedMenu

```
hiCreateVerticalFixedMenu(  
    s_menuHandle  
    l_menuItems  
    x_rows  
    x_cols  
)  
=> r_hiMenu
```

Description

Returns an internal data structure that describes a vertical fixed menu object. The fixed menu is created as a grid of *row* x *column* dimensions, and contains menu item objects created by [hiCreateMenuItem](#). An additional *Done* item is added as the last menu item. Selecting this *Done* menu item will remove the fixed menu from the screen.

The width and height of the vertical fixed menu is determined by the number of rows and columns in the menu. Once this menu is created, you can display it with [hiDisplayFixedMenu](#).

Vertical fixed menus appear along the side of the root window (screen), rather than an application window. These fixed menus are shared among all Cadence application windows contained within the root window. Menu items you want to display within the fixed menu are inserted in row-major order. Keep this in mind when passing arguments to avoid graphically undesirable results.

Arguments

<i>s_menuHandle</i>	A unique global SKILL symbol used to reference this menu.
<i>l_menuItems</i>	List of menu items created by hiCreateMenuItem .
<i>x_rows</i>	Number of rows in the fixed menu.
<i>x_cols</i>	Number of columns in the fixed menu.

Value Returned

r_hiMenu Returns the SKILL structure representing the menu. The *menuHandle* can be referenced by *menu->hiMenuSym*.

Cadence User Interface SKILL Functions Reference

Menus

Example

See the second example of [hiCreateHorizontalFixedMenu](#) for an example of `hiCreateVerticalFixedMenu`.

Cadence User Interface SKILL Functions Reference

Menus

hiDeleteMenu

```
hiDeleteMenu(  
    r_hiMenu  
)  
=> t / nil
```

Description

Destroys the menu given by *hiMenu*, which must be a valid data structure. Individual menu items contained in this menu are also destroyed.

Arguments

r_hiMenu

The menu returned by any of the menu creation routines, except for *hiCreatePulldownMenu*. (Pulldown menus are destroyed by *hiDeleteBannerMenu*.)

Note: To delete window fixed menus, first use *hiRemoveFixedMenu* to remove the fixed menu from the window, and then use *hiDeleteMenu* to destroy the fixed menu

Value Returned

t

Returns *t* if the menu is deleted.

nil

Returns *nil* and an error message if the menu is not deleted. The SKILL handle referencing this menu is reset to *nil*.

Cadence User Interface SKILL Functions Reference

Menus

hiDeleteMenuItem

```
hiDeleteMenuItem(  
    r_hiMenu  
    s_itemHandle  
)  
=> t / nil
```

Description

Deletes the *menuItem* from all instances of the specified *r_hiMenu*.

Arguments

<i>r_hiMenu</i>	Menu created with hiCreateMenu or hiCreatePulldownMenu .
<i>s_itemHandle</i>	SKILL menu item handle specified when the menu item was created with hiCreateMenuItem or hiCreateSliderMenuItem or hiCreateSeparatorMenuItem .

Value Returned

<i>t</i>	Returns <i>t</i> if the menu item is deleted.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu item is not deleted.

Cadence User Interface SKILL Functions Reference

Menus

hiDisableMenuItem

```
hiDisableMenuItem(  
    r_hiMenu  
    s_itemHandle  
    [w_windowId]  
)  
=> t / nil
```

Description

Grays out menu items (*s_itemHandle*) on pull-down, pop-up, or fixed menus (*r_hiMenu*), making them unselectable.

Unless otherwise specified, all menu items are selectable when created.

Arguments

<i>r_hiMenu</i>	Menu created with <code>hiCreateMenu</code> , <code>hiCreatePulldownMenu</code> , <code>hiCreateHorizontalFixedMenu</code> , or <code>hiCreateVerticalFixedMenu</code> .
<i>s_itemHandle</i>	SKILL menu item handle specified when the menu item was created with <code>hiCreateMenuItem</code> or <code>hiCreateSliderMenuItem</code> .
<i>w_windowId</i>	Window argument created with <code>hiCreateWindow</code> or <code>hiOpenWindow</code> (refer to “Window Management Functions”). If this argument is used, only the menu item in a menu on this window is disabled. This is useful for shared menus, when you want to disable a menu item in one menu only and leave the other occurrences of the same menu item enabled. This argument is also necessary when disabling a menu item on a window fixed menu (see hiAddFixedMenu).

Value Returned

<i>t</i>	Returns <i>t</i> if the menu items are disabled.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu items are not disabled.

hiDisplayEdgeMenu

```
hiDisplayEdgeMenu(  
    r_hiMenu  
    [l_menuLocation]  
    [x_width]  
    [x_height]  
)  
=> t / nil
```

Description

Displays an Edge-style popup or fixed menu, created from the *menutrans* program.

Note: Edge-style fixed menus can also be displayed with [hiDisplayFixedMenu](#).

Arguments

<i>r_hiMenu</i>	The Edge-style menu created from the <i>menutrans</i> program. This menu is created after loading the SKILL files generated by <i>menutrans</i> .
<i>l_menuLocation</i>	This optional argument determines the position the menu will be displayed. If specified, this argument must contain a list of 2 integers representing an x,y screen coordinate (assuming 0:0 to be the upper left corner of the screen). The menu will be displayed with its upper left corner at this position. If not specified and <i>r_hiMenu</i> is a popup menu, the menu will be displayed in the center of the screen; if not specified and <i>r_hiMenu</i> is a fixed menu, the menu will be displayed at the position it was created at (see hiSetFixedMenuSize).
<i>x_width</i>	An optional argument for Edge-style fixed menus. If specified, the menu will be displayed with the specified <i>width</i> ; otherwise, if not specified, or set to 0, it will be displayed with the width it was created or translated with.
<i>x_height</i>	An optional argument for Edge-style fixed menus. If specified, the menu will be displayed with the specified <i>height</i> ; otherwise, if not specified, or set to 0, it will be displayed with the height it was created or translated with.

Cadence User Interface SKILL Functions Reference

Menus

Value Returned

<code>t</code>	Returns <code>t</code> if the menu is displayed.
<code>nil</code>	Returns <code>nil</code> and an error message if the menu is not displayed.

hiDisplayFixedMenu

```
hiDisplayFixedMenu(  
    r_hiMenu  
    g_menuLocation  
    [x_width]  
    [x_height]  
)  
=> t / nil
```

Description

Displays the specified fixed menu in the default root window (screen) for the application. They appear on a side of the screen, to be shared among Cadence application windows contained within the screen. This function will display both Opus-style (created from [hiCreateHorizontalFixedMenu](#) or [hiCreateVerticalFixedMenu](#)) and Edge-style fixed menus (created from the *menutrans* program).

Note: Edge-style fixed menus can also be displayed with [hiDisplayEdgeMenu](#).

Arguments

r_hiMenu The menu returned from [hiCreateHorizontalFixedMenu](#) or [hiCreateVerticalFixedMenu](#) **or** an Edge-styled fixed menu created with the *menutrans* program.

g_menuLocation If *r_hiMenu* is an Opus-style menu, this argument must be one of the following to place the fixed menu along the corresponding border of the root window:

```
"top"  
"bottom"  
"left"  
"right"
```

If *r_hiMenu* is an Edge-style menu, this argument is optional. If specified, this argument must contain a list of 2 integers representing an x,y screen coordinate (assuming 0:0 to be the upper left corner of the screen). The fixed menu will be displayed with its upper left corner at this position.

Fixed menus may be stacked one on top of the other; only the topmost fixed menu receives interaction.

Cadence User Interface SKILL Functions Reference

Menus

<i>x_width</i>	Specifies the width of the menu. If you do not specify the width or set it to 0, the menu is displayed with the width it was created with (which is determined by the number of rows and columns specified in hiCreateVerticalFixedMenu or hiCreateHorizontalFixedMenu).
<i>x_height</i>	Specifies the height of the menu. If you do not specify the height or set it to 0, the menu is displayed with the height it was created with (which is determined by the number of rows and columns specified in hiCreateVerticalFixedMenu or hiCreateHorizontalFixedMenu).

Value Returned

<i>t</i>	Returns <i>t</i> if the menu is displayed.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu is not displayed.

Example

The following example displays `trExampleHorizontalFixedMenu`. Before using this example, see the [hiCreateHorizontalFixedMenu](#) example to find out how to create `trExampleHorizontalFixedMenu`.

```
hiDisplayFixedMenu(  
    trExampleHorizontalFixedMenu  
    "top"  
)
```

Cadence User Interface SKILL Functions Reference

Menus

hiDisplayMenu

```
hiDisplayMenu(  
    r_hiMenu  
    [l_position]  
)  
=> t / nil
```

Description

Displays a menu created by `hiCreateMenu`, `hiCreateSimpleMenu`, or `hiCreate2DMenu`. For non-2D menus, the menu is always displayed at the current cursor's position at the time `hiDisplayMenu` is called.

Arguments

<i>r_hiMenu</i>	The menu returned from a call to hiCreateMenu , hiCreateSimpleMenu or hiCreate2DMenu .
<i>l_position</i>	Optional x, y screen coordinates used for 2D menus. If specified, the 2D menus will be displayed with its upper left corner at this position (assuming 0:0 to be the upper left corner of the screen). If not specified, the 2D menu will be displayed in the center of the screen, or at the last position it was displayed.

Value Returned

<i>t</i>	Returns <i>t</i> if the menu is displayed.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu is not displayed.

Example

The following example uses the [hiSetBindKey](#) function to define a bindkey for the Schematics application. The bindkey displays the `trExampleMenu` pop-up menu. For an example of creating this pop-up menu, see [hiCreateSimpleMenu](#).

```
hiSetBindKey( "Schematics"  
    "Shift Ctrl<Btn2Down>(2)"  
    "hiDisplayMenu( trExampleMenu )" )
```

Cadence User Interface SKILL Functions Reference

Menus

hiDisplayWindowMenu

```
hiDisplayWindowMenu(  
    w_windowId  
)  
=> t / nil
```

Description

Displays the popup menu associated with the given *w_windowId*, which must be a valid data structure returned by a call to hiOpenWindow or hiCreateWindow.

Arguments

<i>w_windowId</i>	Window ID you specify. A menu can be associated with a window by calling <u>hiSetWindowMenu</u> . The menu is displayed at the current cursor's position at the time <u>hiDisplayWindowMenu</u> was called. The menu is brought down by a menu selection or by clicking outside the menu.
-------------------	---

Value Returned

t	Returns t if the menu is displayed.
nil	Returns nil and an error message if the menu is not displayed.

hiEdgeFixedMenuDone

```
hiEdgeFixedMenuDone(  
    r_hiMenu  
)  
=> t / nil
```

Description

Removes the specified Edge-style fixed menu from the screen.

Note: This function may be removed in a future release. (See [hiFixedMenuDown](#).)

Arguments

<i>r_hiMenu</i>	Edge-style fixed menu created with <i>menutrans</i> .
-----------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the menu is removed.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu is not removed.

Cadence User Interface SKILL Functions Reference

Menus

hiEnableMenuItem

```
hiEnableMenuItem(  
    r_hiMenu  
    s_itemHandle  
    [w_windowId]  
)  
=> t / nil
```

Description

Makes unselectable menu items (*s_itemHandle*) in pull-down, pop-up, or fixed menus (*r_hiMenu*) selectable (ungrays them).

All menu items are selectable when created.

Arguments

<i>r_hiMenu</i>	Menu created with hiCreateMenu , hiCreatePulldownMenu , hiCreateHorizontalFixedMenu , or hiCreateVerticalFixedMenu .
<i>s_itemHandle</i>	SKILL menu item handle specified when the menu item was created with hiCreateMenuItem or hiCreateSliderMenuItem .
<i>w_windowId</i>	Window argument created with hiCreateWindow or hiOpenWindow (refer to “Window Management Functions”). If this argument is specified, only the menu item in a menu on this window is enabled. This is useful for shared menus, when you want to enable a menu item in one menu only and leave the other occurrences of the same menu item disabled. This argument is also necessary when disabling a menu item on a window fixed menu (see hiAddFixedMenu).

Value Returned

<i>t</i>	Returns <i>t</i> if the menu items are disabled.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu items are not disabled.

hiFixedMenuDown

```
hiFixedMenuDown(  
    r_hiMenu  
)  
=> t / nil
```

Description

Removes the specified fixed menu from the screen. This function should only be used for Edge-style fixed menus; it is automatically placed as the callback for the *Done* menu item on Opus-style fixed menus.

Arguments

<i>r_hiMenu</i>	Edge-style fixed menu created with <i>menutrans</i> or an Opus-style fixed menu created with <u>hiCreateHorizontalFixedMenu</u> or <u>hiCreateVerticalFixedMenu</u> .
-----------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the menu is removed.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu is not removed.

Cadence User Interface SKILL Functions Reference

Menus

hiGetWindowMenu

```
hiGetWindowMenu(  
    w_windowId  
)  
=> s_hiMenu / nil
```

Description

Returns the symbol of the menu associated with the specified window. This menu was associated with a window by using [hiSetWindowMenu](#).

Arguments

<i>w_windowId</i>	Window ID you specify.
-------------------	------------------------

Value Returned

<i>s_hiMenu</i>	Returns the symbol of the menu associated with the window.
<i>nil</i>	Returns <i>nil</i> if there is no menu associated with the window.

Cadence User Interface SKILL Functions Reference

Menus

hiInsertMenuItem

```
hiInsertMenuItem(  
    r_hiMenu  
    r_menuItem  
    g_position  
)  
=> t / nil
```

Description

Inserts the *r_menuItem* into a specified position of all instances of *r_hiMenu*.

Arguments

<i>r_hiMenu</i>	Menu created with hiCreateMenu or hiCreatePulldownMenu .
<i>r_menuItem</i>	Menu item description returned from hiCreateMenuItem , hiCreateSliderMenuItem , or hiCreateSeparatorMenuItem .
<i>g_position</i>	Symbol or integer specifying the item to insert after. The only way to insert a menu item at the beginning of a menu is to specify position 0. An integer position that is larger than the number of menu items defaults to the end of the menu; a negative position defaults to 0. Any symbol specified must represent the menu item handle used when the menu item was created with hiCreateMenuItem or hiCreateSliderMenuItem .

Value Returned

t	Returns t if the menu item is inserted.
nil	Returns nil and an error message if the menu item is not inserted.

Cadence User Interface SKILL Functions Reference

Menus

hIs2DMenu

```
hIs2DMenu(  
    g_menu  
)  
=> non-nil / nil
```

Description

Determines whether the specified argument is a valid 2D menu.

Arguments

<i>g_menu</i>	Menu you specified.
---------------	---------------------

Value Returned

<i>non-nil</i>	Returns a non-nil value if the specified menu is a valid 2D menu.
<i>nil</i>	Returns <i>nil</i> if the menu is not a 2D menu.

Cadence User Interface SKILL Functions Reference

Menus

hiIsIcon

```
hiIsIcon(  
    g_icon  
)  
=> t / nil
```

Description

Determines whether the specified argument is a valid icon.

Arguments

<i>g_icon</i>	Icon you specify.
---------------	-------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the specified icon is valid.
----------	--

<i>nil</i>	Returns <i>nil</i> if the icon is not valid.
------------	--

Cadence User Interface SKILL Functions Reference

Menus

hiIsMenu

```
hiIsMenu(  
    g_menu  
)  
=> non-nil / nil
```

Description

Determines whether the specified argument is a valid menu.

Arguments

<i>g_menu</i>	Menu you specified.
---------------	---------------------

Value Returned

non-nil	Returns a non-nil value if the specified menu is a valid popup, pulldown, or fixed menu.
nil	Returns nil if the menu is not valid.

hiIsMenuItemEnabled

```
hiIsMenuItemEnabled(  
    r_hiMenu  
    s_itemHandle  
    [w_windowId]  
)  
=> t / nil
```

Description

Determines whether a menu item is enabled.

A menu item can be enabled with [hiEnableMenuItem](#) and disabled with [hiDisableMenuItem](#).

Arguments

<i>r_hiMenu</i>	Menu created with <u>hiCreateMenu</u> , <u>hiCreatePulldownMenu</u> , <u>hiCreateHorizontalFixedMenu</u> , or <u>hiCreateVerticalFixedMenu</u> .
<i>s_itemHandle</i>	SKILL menu item handle specified when the menu item was created with <u>hiCreateMenuItem</u> or <u>hiCreateSliderMenuItem</u> .
<i>w_windowId</i>	Optional window argument. If specified, only the menu on this window is looked at. This is useful if the same menu has been placed on more than one menu. This argument is also necessary when disabling a menu item on a window fixed menu (see <u>hiAddFixedMenu</u>).

Value Returned

<i>t</i>	Returns <i>t</i> if the specified menu item is enabled.
<i>nil</i>	Returns <i>nil</i> if the menu or menu item cannot be found, or if the menu item is disabled.

hiSetFixedMenuSize

```
hiSetFixedMenuSize(  
    w_windowId  
)  
=> t / nil
```

Description

Sets the window position and size of an Edge-style fixed menu that has been created in Opus.

The user should position and resize the window containing the fixed menu Cellview before calling this procedure. When the fixed menu is displayed it will be displayed with this size at this position. Be sure to save the cellView after calling this procedure.

Arguments

<i>w_windowId</i>	Window ID of the window containing the fixed menu cellView.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window position and menu size are set.
<i>nil</i>	Returns <i>nil</i> and an error message if the window position or menu size are not set.

hiSetMenu

The `hiSetMenu` function has been replaced by the [hiSetWindowMenu](#) function.

hiSetMenuItemCallback

```
hiSetMenuItemCallback(  
    r_hiMenu  
    s_itemHandle  
    t_callback  
)  
=> t / nil
```

Description

Changes the callback associated with all instances of a specified menu item.

Arguments

<i>r_hiMenu</i>	Menu created with <code>hiCreateMenu</code> , <code>hiCreatePulldownMenu</code> , <code>hiCreateHorizontalFixedMenu</code> , or <code>hiCreateVerticalFixedMenu</code> .
<i>s_itemHandle</i>	SKILL menu item handle specified when the menu item was created with <code>hiCreateMenuItem</code> or <code>hiCreateSliderMenuItem</code> .
<i>t_callback</i>	New SKILL procedure that will be executed when this menu item is selected. Separate multiple procedures with a space.

Value Returned

<i>t</i>	Returns <i>t</i> if the callback is changed.
<i>nil</i>	Returns <i>nil</i> and an error message if the callback is unchanged.

Cadence User Interface SKILL Functions Reference

Menus

hiSetMenuItemText

```
hiSetMenuItemText(  
    r_hiMenu  
    s_itemHandle  
    t_itemText  
)  
=> t / nil
```

Description

Changes the item text associated with all instances of a specified menu item.

Arguments

<i>r_hiMenu</i>	Created with <code>hiCreateMenu</code> or <code>hiCreatePulldownMenu</code> .
<i>s_itemHandle</i>	SKILL menu item handle specified when the menu item was created with <code>hiCreateMenuItem</code> or <code>hiCreateSliderMenuItem</code> .
<i>t_itemText</i>	New item text that the menu item will contain.

Value Returned

<i>t</i>	Returns <i>t</i> if the text is changed.
<i>nil</i>	Returns <i>nil</i> and an error message if the text is unchanged.

Cadence User Interface SKILL Functions Reference

Menus

hiSetWindowMenu

```
hiSetWindowMenu(  
    g_menu  
    w_windowId  
)  
=> t / nil
```

Description

Associates the given *g_menu* with the specified *windowId*.

Arguments

<i>g_menu</i>	Menu must have been returned by a call to either hiCreateSimpleMenu or hiCreateMenu . If menu is <i>nil</i> , any previous menu will be disassociated with the window.
<i>w_windowId</i>	Window ID you specify. You should not associate a fixed menu with a window. Only one menu can be associated with a window at any given time. Multiple calls to this routine disassociate any previous menu that was set. After a menu has been associated with a window, the menu can be displayed with hiDisplayWindowMenu .

Value Returned

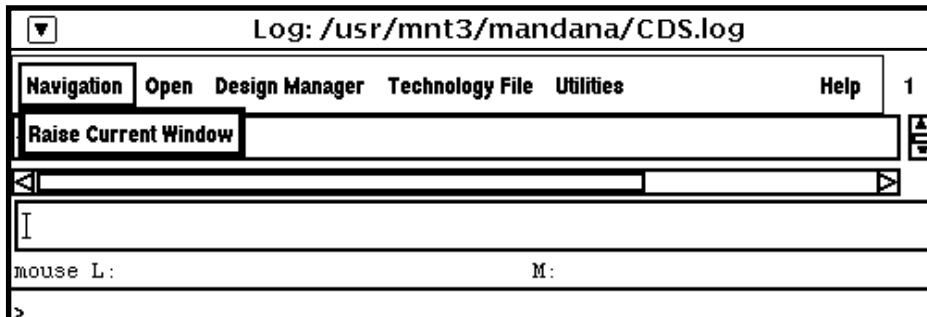
<i>t</i>	Returns <i>t</i> if the menu is set.
<i>nil</i>	Returns <i>nil</i> and an error message if the menu is not set.

Programming Samples

The following samples show you how to create different types of menus.

Creating a Sample Pulldown Menu

The following example, creates a pulldown menu named Navigation and inserts it in the leftmost position of the CIW menu banner. If you select the Navigation-Raise Current Window, the current window will be raised to the top of the screen.



```
;;; Creating a new window to be used as the current window
hiOpenWindow()

;;; creating the "Raise Current Window" menu item
trRaiseCurrentWindow = hiCreateMenuItem(
    ?name      'trRaiseCurrentWindow
    ?itemText   "Raise Current Window"
    ?callback   "hiRaiseWindow( hiGetCurrentWindow() )"
)

;;; creating the Navigation pulldown menu
hiCreatePulldownMenu(
    'trPulldownMenu    ;;; menu handle
    "Navigation"        ;;; menu title
    list( trRaiseCurrentWindow)
    ""                  ;;; empty help string
) ; hiCreatePulldownMenu

;;; inserting the pulldown menu in the CIW
hiInsertBannerMenu(
    window( 1 )
    trPulldownMenu
    0
)
```

Deleting the Pulldown Menu

To remove the Navigation pulldown menu from the CIW menu banner use the following command:

```
hiDeleteBannerMenu( window(1) 0 )
```

Creating a Sample Popup Menu

The following sample, first creates a popup menu that lists the Design FrameWork II windows currently accessible on the desktop, and then sets the F6 function key to display this popup menu.

```
procedure( trDisplayWindowsPopUp()  
  hiDisplayMenu(  
    ;; creating a dynamic popup menu  
    hiCreateSimpleMenu(  
      'trWindowsPopUp  
      "Windows"  
      foreach( mapcar wid hiGetWindowList()  
        hiGetWindowName( wid )  
      ) ; foreach  
      foreach( mapcar wid hiGetWindowList()  
        sprintf(  
          nil  
          "hiRaiseWindow( window( %d ))"  
          wid->windowNum  
        )  
      ) ; foreach  
    ) ; hiCreateSimpleMenu  
  ) ; hiDisplayMenu  
) ; procedure  
  
;; Setting the F6 function key to display the popup menu  
hiSetBindKey( "Command Interpreter" "<Key>F6"  
"trDisplayWindowsPopUp()" )
```

Creating a Sample Fixed Menu

The following sample creates a vertical fixed menu, displays it, and then attaches it to a window. In this example, the `trCreateMenuItem` procedure allows an easy means of generating menu items with a similar callback.

```
;;; The following procedure returns a menu item
;;; based on theMenuSymbol.
procedure( trCreateMenuItem( theMenuSymbol )
  ;;; The set function assigns the menu item data structure
  ;;; to the theMenuSymbol's value.
  set(
    theMenuSymbol
    hiCreateMenuItem(
      ?name      theMenuSymbol
      ;;; get_pname returns the name of the symbol
      ;;; as a text string.
      ?itemText  get_pname( theMenuSymbol )
      ;;; All menu items will have a similar callback
      ;;; that prints the menu item's ?name variable.
      ?callback  sprintf( nil "println( '%L )" theMenuSymbol )
    )
  ) ; set
) ; procedure

;;; creating the menu items for the fixed menu
trCreateMenuItem( 'item1 )
trCreateMenuItem( 'item2 )
trCreateMenuItem( 'item3 )

;;; creating the vertical fixed menu
hiCreateVerticalFixedMenu(
  'trExampleVerticalFixedMenu
  list( item1 item2 item3 )
  3      ;;; number of rows
  1      ;;; number of columns
)
```

Displaying a Fixed Menu

To display the sample fixed menu, use the following function:

```
hiDisplayFixedMenu(
  trExampleVerticalFixedMenu
  "left"    ;;; placement must be one of "top", "bottom",
            ;;; "right", or "left"
)
```

After the fixed menu is displayed, press the menu items. Each menu item displays its name in the CIW.

Press the Done button to close to close the fixed menu.

Cadence User Interface SKILL Functions Reference

Menus

Attaching the Fixed Menu to a Window

You can add a vertical fixed menu to a window of type `graphics`, `browser`, `form`, `text`, or `encap`. The following example opens a text file named `fixedmenu.sample` and attaches the `trExampleVerticalFixedMenu` to this text window. You can open a different window and replace its `wid` with `window(3)` in the following example.

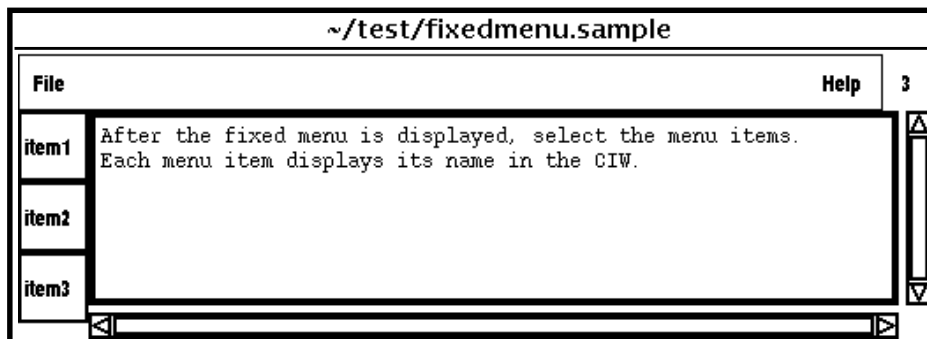
```
view("~/test/fixedmenu.sample")  
;;; The file is displayed as window(3).
```

```
hiAddFixedMenu( ?window window(3) ?fixedMenu trExampleVerticalFixedMenu)
```

After the fixed menu is attached to the window, select the menu items. Each menu item displays its name in the CIW.

Removing a Fixed Menu

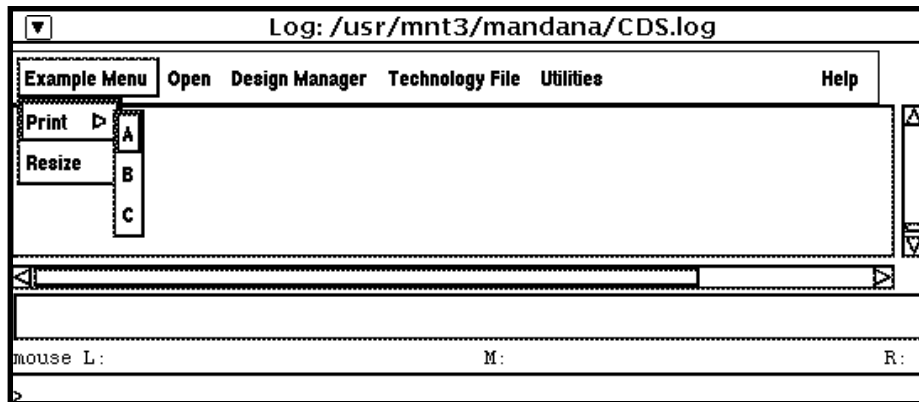
To remove `trExampleVerticalFixedMenu` from `window(3)`, use the following command:



```
hiRemoveFixedMenu( window(3) )
```

Creating a Sample Slider Menu

The following sample creates a pulldown menu named Example Menu. The first menu item of this pulldown menu is a slider menu item that prints the letters A, B, or C when selected. The second menu item resizes the CIW.



```
;;; creating menu items for the slider menu
trA_MenuItem = hiCreateMenuItem(
    ?name 'trA_MenuItem
    ?itemText "A"
    ?callback "println('A)" ;;; prints A in the CIW
)

trB_MenuItem = hiCreateMenuItem(
    ?name 'trB_MenuItem
    ?itemText "B"
    ?callback "println('B)" ;;; prints B in the CIW
)

trC_MenuItem = hiCreateMenuItem(
    ?name 'trC_MenuItem
    ?itemText "C"
    ?callback "println('C)" ;;; prints C in the CIW
)

;;; building the slider menu
hiCreatePulldownMenu(
    'trSubMenu
    ""
    list( ; the list of menu items in the slider menu
        trA_MenuItem
        trB_MenuItem
        trC_MenuItem
    )
)

;;; creating the first menu item for the pulldown menu
trSliderMenuItem = hiCreateSliderMenuItem(
    ?name 'trSliderMenuItem
    ?itemText "Print"
    ?subMenu trSubMenu
)
```


Cadence User Interface SKILL Functions Reference

Menus

```
;;; creating the second menu item for the pulldown menu
trMenuItemResize = hiCreateMenuItem(
    ?name 'trMenuItemResize
    ?itemText "Resize"
    ?callback "hiResizeWindow( window(1) list(0:0 500:500))"
)

;;; creating the Example Menu pulldown menu
hiCreatePulldownMenu(
    'trPulldownMenu
    "Example Menu"
    list( trSliderMenuItem trMenuItemResize )
)

;;; inserting the pulldown menu in the CIW
hiInsertBannerMenu( window(1) trPulldownMenu 0 )
```

Testing the Sample Menu

To test the sample menu, from the Example Menu pulldown menu, select Print. Selecting each letter in the slider menu will display it in the CIW. Selecting the Resize menu item from the pulldown menu will resize the CIW.

Removing the Pulldown Menu

To delete the Example Menu, use the following command:

```
hiDeleteBannerMenu( window(1) 0 )
```

Adding an Entry to a Menu

The following code adds a new entry, *Library Browser*, to the CIW. It looks for the *Library Manager* entry in all the menus of the CIW, and adds an entry for the Library Browser immediately after it.

```
let( ((ciwBannerMenus hiGetBannerMenus(window(1))) toolMenu
      pdItem menuList found libBrowser)
  while(ciwBannerMenus
    pdItem = eval(car(ciwBannerMenus))
    ciwBannerMenus = cdr(ciwBannerMenus)
    menuList = pdItem ->_menuItemList
    while(menuList
      menuItem = car(menuList)
      menuList = cdr(menuList)
      when(menuItem == 'LibMan
        found = t
        menuList = nil
        ciwBannerMenus = nil
      )
    )
  )
  when( found
    libBrowser = hiCreateMenuItem(
      ?name          'LibBrowser
      ?itemText      "Library Browser..."
      ?callback      "dmbOpenLibDAGBrowser()"
    )
    hiInsertMenuItem(pdItem libBrowser 'LibMan)
  )
)
```

Dialog Boxes

The following topics are discussed in this chapter:

- [Overview](#) on page 171
- [Programming Samples](#) on page 183

Overview

Dialog boxes are pop-up windows containing a message that is displayed to the user. Each dialog box has three buttons: *OK*, *Cancel* and *Help*. They can only be closed by pressing the *OK* or *Cancel* button. The dialog box can also be closed by entering *Return* while the cursor is within the box. This has the same effect as pressing the *OK* button.

Note: Do not use the window manager to close dialog boxes or forms. Close or cancel dialog boxes with the buttons in the dialog box.

There are two main types of dialog boxes:

- Modal, or blocking, dialog boxes
- Modeless, or non-blocking, dialog boxes

A modal dialog box waits for your acknowledgment before the application continues. To continue in your application and remove the dialog box from the screen, you must confirm or cancel your request. There are two types of modal dialog boxes:

- Application modal
- System modal

The application modal dialog box will not let you continue with any Cadence application until you acknowledge it. However, you will still be able to execute window manager commands, and any other response to applications that may be running.

The system modal dialog box will not let you do anything until it has been acknowledged. This includes accessing any other application that may be running, the Help button, and the

Cadence User Interface SKILL Functions Reference

Dialog Boxes

context-sensitive help. When a system modal dialog box is displayed, the cursor changes shape, except when it is within the dialog box.

A modeless dialog box is often used as an information box. It is usually displayed by an application to alert you with additional information or give a warning message. The application may proceed, but the dialog box will remain on your screen until you dismiss it.

You can use the routines defined below to create and display a dialog box. When you acknowledge the dialog box by selecting *OK* or *Cancel* or by pressing *Return*, it is removed from the screen.

All dialog boxes require the same arguments.

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDBoxCancel

```
hiDBoxCancel(  
    g_dboxID  
    [g_cancelFromCancel]  
)  
=> t / nil
```

Description

Function logged when either the *Cancel* or the *No* button is pressed on a dialog box. This function can also be called directly from SKILL to simulate pressing *No* or *Cancel*.

If *g_cancelFromCancel* is non-nil in the logged entry, that indicates that the *Cancel* button was pressed when both *No* and *Cancel* buttons are present in the dialog box.

Arguments

g_dboxID ID of the dialog box being canceled.

[*g_cancelFromCancel*] Optional argument to be used when a form has both a *No* and *Cancel* button. The valid values for this argument are:

Any non-nil value: Simulate pressing *Cancel* if the dialog box has a *No* button.

nil: Simulate pressing *No* if the dialog box has both *No* and *Cancel* buttons, otherwise simulate pressing *Cancel*.

The default value of this argument is *nil*.

Value Returned

nil Returns *nil* if the *g_dboxID* is invalid.

t Returns *t* otherwise.

Reference

[hiDBoxOK](#), [hiDisplayAppDBox](#)

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDBoxOK

```
hiDBoxOK(  
    g_dboxID  
    [g_dontUnmanage]  
)  
=> t / nil
```

Description

Function logged when the *OK*, *Yes* or *Done* button is pressed in a dialog box. Can be called directly from SKILL to simulate pressing *OK*, *Yes* or *Done*.

The *g_dontUnmanage* optional argument should not be used, as it is only intended for special quick help dialog boxes that are currently unused.

Arguments

<i>g_dboxID</i>	ID of the dialog box.
[<i>g_dontUnmanage</i>]	Do not use this argument.

Value Returned

<i>t</i>	Returns <i>t</i> if the <i>OK</i> , <i>Yes</i> , or <i>Done</i> button is pressed or an <i>hiDBoxOK</i> call is made from SKILL.
<i>nil</i>	Returns <i>nil</i> otherwise.

Reference

[hiDBoxCancel](#), [hiDisplayAppDBox](#)

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDisplayAppDBox

```
hiDisplayAppDBox(  
    ?name s_dboxHandle  
    [?dboxBanner t_dboxBanner]  
    [?dboxText t_dboxText]  
    [?callback t_callback]  
    [?dialogType x_dialogType]  
    [?dialogStyle s_dialogStyle]  
    [?buttonLayout s_buttonLayout]  
    [?defaultButton x_defaultButton]  
    [?location l_location]  
    [?help t_help]  
)  
=> t / nil / cancel
```

Description

Creates and displays a dialog box. The dialog box is destroyed when it is removed from the screen.

Arguments

<i>s_dboxHandle</i>	SKILL handle to the dialog box. The <i>dboxHandle</i> symbol is set to point to the created dialog box; when the dialog box is dismissed, this symbol is reset to <i>nil</i> .
<i>t_dboxBanner</i>	Text that appears within the window manager banner of the dialog box.
<i>t_dboxText</i>	Message that is displayed in the dialog box.
<i>t_callback</i>	Text string specifying the SKILL procedure to be executed when the user selects <i>OK</i> or <i>Yes</i> in the dialog box.
<i>x_dialogType</i>	<p>The following dialog box types, which determine the <i>look</i> of the dialog box, are supplied:</p> <pre>hicWarningDialog default hicErrorDialog hicInformationDialog hicMessageDialog hicQuestionDialog hicWorkingDialog</pre>

Cadence User Interface SKILL Functions Reference

Dialog Boxes

<i>s_dialogStyle</i>	Symbol that represents the blocking behavior of this dialog box. Acceptable choices are 'modal, 'modeless, or 'systemModal. The default blocking behavior of dialog boxes is modal, where the Cadence application is suspended until the dialog box is responded to. systemModal dialog boxes block all applications in the system and do not allow the user to access the help button and/or context-sensitive help until the dialog box is responded to. Modeless dialog boxes do not block any applications.
<i>s_buttonLayout</i>	Symbol specifying the buttons that appear in this dialog box. Possible choices are 'OKCancel (default), 'YesNo, 'YesNoCancel, 'CloseHelp, and 'Close. A <i>Help</i> button appears with all choices except 'Close. The dialog box callback, if any, is executed only if the buttonLayout is not of type 'Close or 'CloseHelp, which simply dismiss the dialog box with no action taken.
<i>x_defaultButton</i>	Specifies the default button to be used when the <i>Return</i> key is pressed. The default button should be specified by its position. Possible choices can range from 1 through 4 depending on <i>s_buttonLayout</i> . For example, if the <i>s_buttonLayout</i> argument is set to 'YesNo, and the <i>x_defaultButton</i> is set to 2, the No button will be the default button. If no value or an invalid value (such as "3" when there are only two buttons) is given for this argument, the first button will be the default.
<i>l_location</i>	The xy coordinates of the upper left corner of the box.
<i>t_help</i>	Text string, containing one keyword, which indexes into a FrameMaker help document. This document will be displayed when the user selects the <i>Help</i> button.

Note: This argument is currently unused.

Value Returned

<i>t</i>	Returns <i>t</i> if you select the <i>OK</i> , <i>Yes</i> , or <i>Close</i> button. If <i>s_dialogStyle</i> is 'modeless, the function returns <i>t</i> immediately after the dialog box is displayed, if there is no error; it does not wait for a button to be pressed.
<i>nil</i>	Returns <i>nil</i> if you select the <i>Cancel</i> or <i>No</i> button, if an error occurs, or if one of the arguments is invalid.

Cadence User Interface SKILL Functions Reference

Dialog Boxes

<code>'cancel</code>	Returns the symbol <code>'cancel</code> if the <i>Cancel</i> button is pressed and <i>s_buttonLayout</i> is <code>'YesNoCancel</code> .
----------------------	---

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDisplayBlockingDBox

hiDisplayBlockingDBox()

Description

hiDisplayAppDBox is the recommended dialog box interface. Please use that function instead of this one.

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDisplayModalDBox

```
hiDisplayModalDBox(  
    s_dBoxHandle  
    t_dBoxBannerText  
    t_dBoxMsg  
    t_dBoxHelp  
    t_confirmAction  
    [l_location]  
    [x_dboxType]  
)  
=> t / nil
```

Description

hiDisplayAppDBox is the recommended dialog box interface. Please use that function instead of this one.

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDisplayModelessDBox

```
hiDisplayModelessDBox(  
    s_dBoxHandle  
    t_dBoxBannerText  
    t_dBoxMsg  
    t_dBoxHelp  
    t_confirmAction  
    [l_location]  
    [x_dboxType]  
)  
=> t / nil
```

Description

hiDisplayAppDBox is the recommended dialog box interface. Please use that function instead of this one and set *?dialogStyle* to 'modeless.

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDisplayNonBlockingDBox

hiDisplayNonBlockingDBox()

Description

hiDisplayAppDBox is the recommended dialog box interface. Please use that function instead of this one.

Cadence User Interface SKILL Functions Reference

Dialog Boxes

hiDisplaySysModalDBox

```
hiDisplaySysModalDBox(  
    s_dBoxHandle  
    t_dBoxBannerText  
    t_dBoxMsg  
    t_dBoxHelp  
    t_confirmAction  
    [l_location]  
    [x_dboxType]  
)  
=> t / nil
```

Description

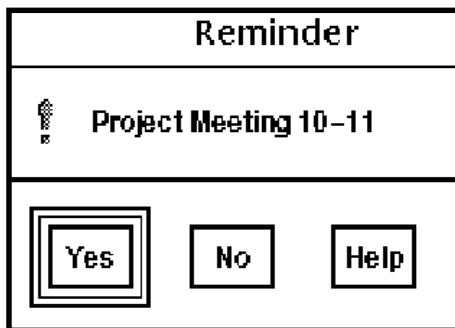
hiDisplayAppDBox is the recommended dialog box interface. Please use that function instead of this one and set *?dialogStyle* to 'systemModal.

Programming Samples

The following samples show you how to create different types of dialog boxes.

Creating a Sample Modeless Dialog Box

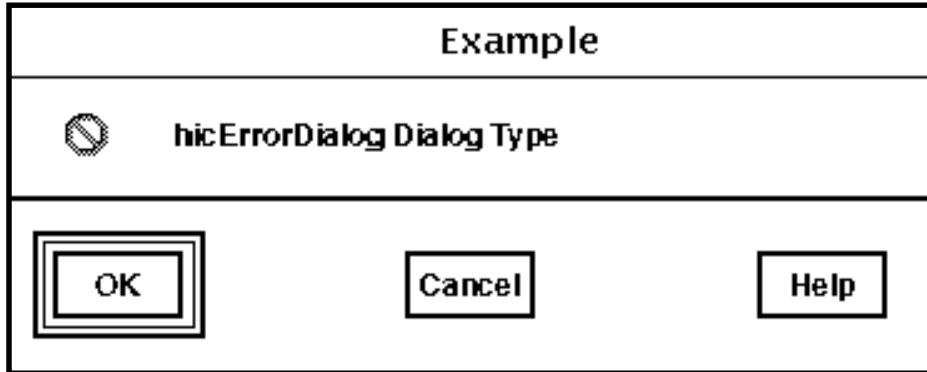
The following sample creates a reminder list and a warning dialog box for each item of the list.



```
;;; creating the reminder list
trReminders = '( "Project Meeting 10-11" "Workout 12-1" "Team Meeting 3-4" )
foreach( reminderText trReminders ; for each reminder
  hiDisplayAppDBox(
    ?name      gensym( 'trReminderDialogBox ) ; unique variable
    ?dboxBanner "Reminder"
    ?dboxText  reminderText
    ?callback  sprintf( nil "trReminderCB( \"%s\" )"  reminderText ) ;
dynamically built callback
    ?dialogType  hicWarningDialog
    ?dialogStyle 'modeless
    ?buttonLayout 'YesNo
  )
) ; foreach
procedure( trReminderCB( reminderText )
  printf( "%s completed" reminderText )
)
```

Creating a Sample Modal Dialog Box

The following sample creates different types of modal dialog boxes.



```
trDialogTypes = '(
  hicWarningDialog hicErrorDialog hicInformationDialog
  hicMessageDialog hicQuestionDialog hicWorkingDialog
)

trDialogTypes = listVariables( "^hic.*Dialog$" )
foreach( dialogTypeSym trDialogTypes
  hiDisplayAppDBox(
    ?name gensym( 'trExampleDialogBox )
    ?dboxBanner "Example"
    ?dboxText sprintf( nil "%L Dialog Type" dialogTypeSym)
    ?dialogType symeval( dialogTypeSym )
    ?dialogStyle 'modal
  )
) ; foreach
```

List Boxes

The following topics are discussed in this chapter:

- [Overview](#) on page 185
- [Programming Samples](#) on page 196

Overview

List boxes display a list of user-defined strings in a scrollable window. The window has a vertical and horizontal scrollbar and the size of the window is constant. You can select a string (or strings) using the left mouse button.

The string(s) that you select can be used by the application after the window is dismissed. Both the `Cancel` and `OK` buttons remove the window from the screen; however, using the `Cancel` button ignores your selections.

The following routines are used to define and display a list box, and can programmatically remove them from the screen.

Note: Use the routine `hiShowListBox` rather than `hiDisplayListBox`.

The selection mechanism for multiple selection in list boxes (using `hiShowListBox`) is as follows:

To select a continuous range of items:

1. Press the left mouse button.
2. Drag the cursor over the items you wish to select.
3. Release the mouse button.

Each time the left mouse button is pressed, a new selection process begins, and the previously selected items are unselected.

To modify a selection:

Cadence User Interface SKILL Functions Reference

List Boxes

1. Press and hold the *Shift* key.
2. Move to the new endpoint of selected items.
3. Press the left mouse button.

The items between the initial start point and the new endpoint are selected; any previous selections are not unselected.

To add or delete items from the selected set:

1. Press and hold the *Control* key.
2. Move to the item you wish to add or delete.
3. Press the left mouse button (will invert the selection state of this item).
4. Drag the mouse button, if you wish to add or delete a range of items.

The item(s) selected or unselected will be added to or deleted from the selected set. Any previous selections remain unaffected.

Cadence User Interface SKILL Functions Reference

List Boxes

hiDisplayListBox

```
hiDisplayListBox(  
    s_listBoxHandle  
    t_listBoxTitle  
    l_listItems  
    t_help  
)  
=> t / nil
```

Description

Creates and displays a list box. The list box will display a list of strings. The user can select exactly one of these strings by clicking over the item with the left mouse button. The list box remains on the screen until the user selects *OK* or *Cancel*. The list box is destroyed when it is removed from the screen

Note: This function will be removed in a future release—use [hiShowListBox](#) instead.

Arguments

<i>s_listBoxHandle</i>	A unique global SKILL handle to the created list box. The selected item can be retrieved from the <i>listBoxHandle</i> . For example, if <i>listBoxHandle</i> was <code>myListBox</code> , the selected item would be <i>myListBox -> value</i> This value will be a list of two items: the integer representing the position of the selected item in the item list and the selected item string.
<i>t_listBoxTitle</i>	Title of the list box displayed in the <i>WindowManager</i> banner.
<i>l_listItems</i>	List of strings that is displayed in the scrolling window.
<i>t_help</i>	String containing information about this list box.

Value Returned

<i>t</i>	Returns <i>t</i> if the user selects <i>OK</i> .
<i>nil</i>	Returns <i>nil</i> if <i>Cancel</i> is selected.

hiGetListBoxFieldFit

```
hiGetListBoxFieldFit(  
    x_numRows  
    x_numColumns  
    t_font  
)  
=> l_widthHeight
```

Description

Returns the width and height necessary for displaying a list box with the specified number of rows and columns and the specified font.

Arguments

<i>x_numRows</i>	Number of rows in the list box.
<i>x_numColumns</i>	Number of columns (characters) in the list box.
<i>t_font</i>	Font for the list box.

Value Returned

<i>l_widthHeight</i>	List of the width and height of the list box, allowing for scrollbars and margins. Note that the width, for variable width fonts, is based on the size of a numeric character.
----------------------	--

Example

This example creates a form with a list box field large enough to show four rows of values and as many columns as the largest value in the value list.

```
let( (lbValues (lbColumns 0) lbSize (lbPrompt "My List") lbPromptSize  
    (lbFont hiGetFont("text")))  
    lbValues = list( "this is my first value string"  
                    "this is my second value string"  
                    "this is my third value string"  
                    "this is my fourth value string"  
                    "this is my fifth value string"  
                    "this is my sixth value string"  
                    "this is my seventh value string"  
                    "this is my eighth value string" )  
    foreach( x lbValues  
        let( (y)
```

Cadence User Interface SKILL Functions Reference

List Boxes

```
        y = strlen( x )
        when( y > lbColumns lbColumns = y )
    )
)
lbPromptSize = hiGetTextWidth( lbFont lbPrompt )
lbSize = hiGetListBoxFieldFit( 4 lbColumns lbFont )
MyListBoxField = hiCreateListBoxField(
    ?name 'MyListBoxField
    ?prompt lbPrompt
    ?choices lbValues )
MyListBoxForm = hiCreateAppForm(
    ?name 'MyListBoxForm
    ?fields list( list( MyListBoxField 5:5
                        list( car(lbSize) + lbPromptSize
                            cadr(lbSize))
                            lbPromptSize ) )
    )
)
```

Reference

[hiDisplayListBox](#), [hiGetNumVisibleItems](#)

Cadence User Interface SKILL Functions Reference

List Boxes

hiGetNumVisibleItems

```
hiGetNumVisibleItems(  
    o_listbox  
)  
=> x_count / nil
```

Description

Returns the number of visible items for a list box field.

The field must have been instantiated for this function to return other than `nil`.

Arguments

<code>o_listbox</code>	Name of the list box.
------------------------	-----------------------

Value Returned

<code>x_count</code>	Number of visible items.
----------------------	--------------------------

<code>nil</code>	Returns <code>nil</code> if the field is not instantiated.
------------------	--

Reference

[hiGetListBoxFieldFit](#)

hiListBoxCancel

```
hiListBoxCancel(  
    o_listBox  
)  
=> t / nil
```

Description

Performs the same action as the *Cancel* button on the listbox.

If the listbox is mapped, it will be unmapped. The listbox callback will be called with a first argument of `nil` to indicate that *Cancel* was pressed (see [hiShowListBox](#)).

Arguments

<code>o_listBox</code>	SKILL handle to the listbox. This is the value of the <code>listBoxHandle</code> symbol passed into <code>hiShowListBox</code> .
------------------------	--

Value Returned

<code>t</code>	Returns <code>t</code> if the list box is canceled.
<code>nil</code>	Returns <code>nil</code> if the list box is not canceled.

Cadence User Interface SKILL Functions Reference

List Boxes

hiListBoxDone

```
hiListBoxDone(  
    o_listBox  
)  
=> t / nil
```

Description

Performs the same action as the *OK* button on the listbox.

If the listbox is mapped, it will be unmapped. The listbox callback will be called with a first argument of *t* to indicate that *OK* was pressed (see [hiShowListBox](#)).

Arguments

<i>o_listBox</i>	SKILL handle to the listbox. This is the value of the <i>listBoxHandle</i> symbol passed into <i>hiShowListBox</i> .
------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the list box is unmapped.
<i>nil</i>	Returns <i>nil</i> if the list box is not unmapped.

Cadence User Interface SKILL Functions Reference

List Boxes

hiShowListBox

```
hiShowListBox(  
    ?name s_listBoxHandle  
    ?choices lt_listItems  
    ?callback s_callback  
    [?title t_listBoxTitle]  
    [?multipleSelect g_multiSelect]  
    [?value lt_selectedItem]  
    [?appData g_appData]  
    [?applyButton g_applyButton]  
)  
=> t / nil
```

Description

Creates and displays a list box. The list box displays a list of strings. The list box can be created so that one item is selectable or multiple items are selectable. While the list box is displayed on the screen, the user may change the items in the list box, or change the item(s) selected. The list box remains on the screen until the user selects *OK* or *Cancel*. The list box is destroyed when it is removed from the screen.

Arguments

<i>s_listBoxHandle</i>	A unique global SKILL handle to the created list box. The selected item and other useful information can be retrieved from the <i>listBoxHandle</i> . For example, if <i>listBoxHandle</i> was 'myListBox, the following could be accessed and/or set:
------------------------	---

myListBox->??	;all list box properties (read-only)
myListBox->hiListBoxSym	;s_listBoxHandle (read-only)
myListBox->title	;list box title (read-only)
myListBox->value	;selected item(s)
myListBox->choices	;item(s) displayed in list box
myListBox->appData	;application data

<i>lt_listItems</i>	List of strings that is displayed in the scrolling window. This list can be referenced or reset through:
---------------------	---

listBoxHandle->choices

When the list box choices have been changed, any item(s) previously selected will be unselected (for example, *listBoxHandle->value* will be nil).

Cadence User Interface SKILL Functions Reference

List Boxes

s_callback

The user-defined SKILL procedure that will be executed when either the *OK* or *Cancel* button of the list box is pressed. This procedure must be in the following format:

```
myListBoxCB(g_OKpressed o_listBox)
```

where *g_OKpressed* is *t* if the user pressed the *OK* button and is *nil* if the user pressed *Cancel*. The argument *o_listBox* is the SKILL structure of the list box that was dismissed. Various elements can be accessed through this list box structure, as described above.

t_listBoxTitle

Optional title of the list box displayed in the *WindowManager* banner. This title can also be referenced through:

```
listBoxHandle->title
```

g_multiSelect

If set to *t*, multiple items in the list box can be selected; if set to *nil*, at most one item can be selected at a time. The default value is *nil*.

lt_selectedItem

List of string item(s) that will appear already selected when the list box is displayed. These strings must also be specified in *l_listItems*. If not specified, nothing is pre-selected. If this list contains more than one string, the argument *g_multiSelect* must be set to *t*. The currently selected item(s) can be referenced **or** set through:

```
listBoxHandle->value
```

g_appData

Used to store any application-specific data associated with the list box. It can be referenced **or** set through:

```
listBoxHandle->appData
```

g_applyButton

If set to *t*, places an *Apply* button in the list box. The default is *nil* (no *Apply* button).

There is also an *applying* property which, if queried during the callback of the list box (via `myListBox->applying`), returns *t* if the *Apply* button was the reason for the callback, or *nil* if the callback was called because the user clicked *OK*.

Cadence User Interface SKILL Functions Reference

List Boxes

Value Returned

- `t` Returns `t` if the user selects *OK*.
- `nil` Returns `nil` if *Cancel* is selected.

The list box symbol is reset to `nil` after the callback has been executed, and the list box is dismissed from the screen.

Example

Create and display a list box showing various flowers. The user should be able to select all of his/her favorite flowers.

```
hiShowListBox(?name 'myLB
?title "Pick your favorite flowers"
?choices list("daffodil" "lily" "freesia" "rose" "carnation")
?value list("rose" "freesia")
?callback 'myListBoxCB
?multipleSelect t )

; This procedure is called when the user selects OK or
; Cancel on the list box
procedure(myListBoxCB(okPressed listbox "go")
    ; if user pressed OK, save the selected flowers
    if( okPressed saveFlowers(listbox->value) )
)

; Change the list of flowers being displayed
myLB->choices = list("petunia" "tiger lily"
"bird of paradise" "chrysanthemum")

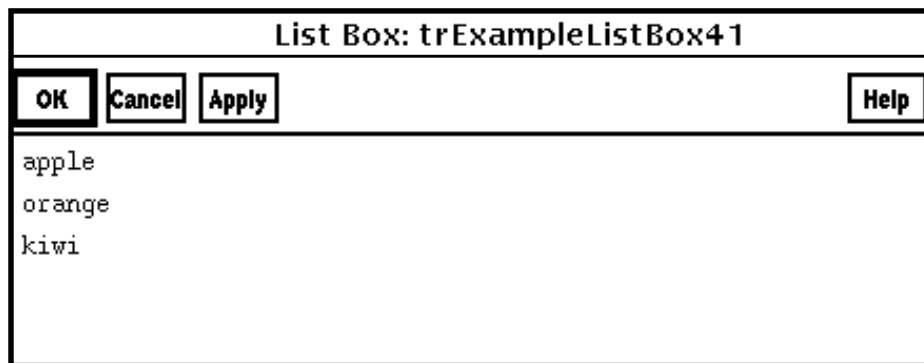
; Select a flower
myLB->value = list("bird of paradise")
```

Programming Samples

The following samples show you how to create list boxes.

Creating a Simple List Box Application

The following sample creates a simple list box.



```
;;; trShowListBox generates a new list box displaying aList
procedure( trShowListBox( aList )
  let( ( listBoxSymbol )
    listBoxSymbol = gensym( 'trExampleListBox )
    hiShowListBox(
      ?name listBoxSymbol
      ?choices aList
      ?callback 'trExampleListBoxCB
      ?title      sprintf( nil "List Box: %L"  listBoxSymbol )
      ?multipleSelect t
      ?applyButton t
    )
  ) ; let
) ; procedure

procedure( trExampleListBoxCB( ok theListBox )
  "The callback routine"
  println( list( ok theListBox->value ) )
) ; procedure
```

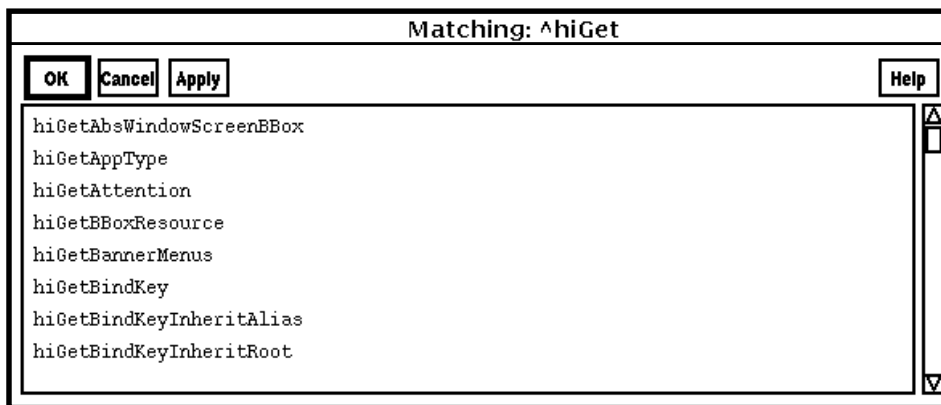
Testing Your Application

To test your application, create two different list boxes and verify that the two list boxes behave independently:

```
trShowListBox( '( "apple" "orange" "kiwi" ) )
trShowListBox( '( "wood" "water" "metal" ) )
```

Creating an Advanced List Box Application

The following sample creates a list box that displays all the functions matching the specified pattern.



```

procedure( trFunctionListBox( pattern )
    "uses trListFunctions to compose a Show List Box"
    hiShowListBox(
        ?name gensym( 'trFunctionListBoxHandle )
        ?title sprintf( nil "Matching: %s" pattern )
        ?choices trListFunctions( pattern )
        ?callback 'trFunctionListBoxCB
        ?applyButton t
        ?multipleSelect t
    ) ; hiShowListBox
    ) ; procedure

procedure( trListFunctions( pattern )
    "returns a list of functions for the matching pattern"
    foreach( mapcar fun listFunctions( pattern )
        get_pname( fun )
    )
    ) ; procedure

procedure( trFunctionListBoxCB( ok theListBox )
    let( ( theArglist )
        when( ok
            foreach( funName theListBox->value
                theArglist = arglist(makeSymbol(funName))
                if( theArglist
                    then
                        printf("\n%s%L" funName theArglist)
                    else
                        printf( "\n%s()" funName )
                ) ; if
            ) ; foreach
        ) ; when
    ) ; let
    ) ; procedure

```

Cadence User Interface SKILL Functions Reference

List Boxes

Testing Your Application

To find all the functions that start with "hiGet", type the following command in the CIW:

```
trFunctionListBox( "^hiGet" )
```

Icons

This chapter describes functions used for creating icons.

Overview

The following icon procedures can create icons from SKILL through an ASCII string description. These icons can be used in forms and menus.

Each pixel of an icon is described by an ASCII alphabetic character (a-z or A-Z) or as one of the special characters '.', '0', or '1'. Each of these characters (except '0' and '1') can be set to a different color, giving the user 27 color possibilities per icon pixel (the characters '0' and '1' are hardcoded to white and black, respectively).

This is not a preferred method of creating icons; this interface will probably change in the future. The preferred method is to create an icon from a graphic representation using `geCellViewToDlist` and `dlDlistToIcon`.

colorIndex

```
colorIndex(  
    l_rgb )  
=> x_index
```

Description

Returns the pixel value for the colormap index of the specified RGB triplet. Alias for [hiMatchColor](#).

hiCreateColorArray

```
hiCreateColorArray(  
    )  
=> a_iconColorArray
```

Description

Returns a 27-element color array that can be used in a call to [hiStringToIcon](#). The first 26 items correspond to the letters in the alphabet; the 27th item corresponds to a period (.). Example: `colorArray[0]` is represented by a or A; `colorArray[12]` is represented by m or M; `colorArray[26]` is represented by a period (.).

To set these array elements, use [hiMatchColorByName](#) or [hiMatchColor](#).

Arguments

None.

Value Returned

a_iconColorArray Returns a 27-element SKILL array.

hiMatchColor

```
hiMatchColor(  
    l_RGBcolor  
)  
=> x_colormapIndex
```

Description

Returns the pixel value for the colormap index of the layer color (currently in the colormap) that most closely matches the specified RGB triplet. The index can change with each invocation of Cadence software and should not be hard coded into any SKILL program.

The pixel value can be used as input for drawing routines that require it, including icon routines and some display list functions.

If the software is running in black-and-white mode, the pixel value will be either 0 or 1. If it is running in color mode, the pixel value will be in the range 2-254 and will be divisible by two, and if the selectionPlane is set to True, the value will be divisible by four.

The function will always return a legal pixel value. If there are no colors that are an exact match, the actual color returned could be quite different from what you expect, as the formula for “closest match” does not always result in what your eyes tell you is the closest match.

Arguments

<i>l_RGBcolor</i>	List of three integers, range 0-1000, specifying the red, green and blue values in the RGB color space for the color.
-------------------	---

Value Returned

<i>x_colormapIndex</i>	Returns the colormap index of the matching layer color. One use of this value is as an element in the array created by hiCreateColorArray .
------------------------	---

Example

This example finds an already allocated layer color that matches most closely to pure red.

```
MyRed = hiMatchColor( list( 1000 0 0 ) )
```

This example finds an already allocated layer color that matches most closely to pure cyan.

Cadence User Interface SKILL Functions Reference

Icons

```
MyCyan = hiMatchColor( list( 0 1000 1000 ) )
```

This example finds an already allocated layer color that matches most closely to the color coral.

```
MyCoral = hiMatchColor( list( 1000 500 314 ) )
```

hiMatchColorByName

```
hiMatchColorByName(  
    t_colorName  
)  
=> x_colormapIndex
```

Description

Returns the pixel value for the colormap index of the layer color (currently in the colormap) that most closely matches the color name supplied. For more information, refer to the *Cadence Design Framework II Configuration User Guide*.

Arguments

<i>t_colorname</i>	A string specified in <code>/usr/lib/X11/rgb.txt</code> .
--------------------	---

Value Returned

<i>x_colormapIndex</i>	Returns the colormap index of the matching layer color, to be used as an element in the array created by hiCreateColorArray . The returned value can range from 0 to 255.
------------------------	---

hiStringToIcon

```
hiStringToIcon(  
    a_colorArray  
    t_iconString  
    x_width  
    x_height  
)  
=> l_hiIcon / nil
```

Description

Returns an internal icon description that can be used in calls to `hiCreatePulldownMenu`, `hiCreateMenuItem`, `hiCreateSliderMenuItem`, `hiCreateButton`, `hiCreateCyclicField`, `hiCreateLabel`, and `hiAddCyclicChoice`.

Arguments

<i>a_colorArray</i>	Array created by hiCreateColorArray . Each element of this array describes a pixel color to use within the icon.
<i>t_iconString</i>	ASCII description of the icon, of <i>x_width</i> width and <i>x_height</i> height. The maximum string length for <i>t_iconString</i> is 8K (8192) characters.
<i>x_width</i>	Width of the icon.
<i>x_height</i>	Height of the icon.

Value Returned

<i>l_hiIcon</i>	Returns a list describing the icon.
<i>nil</i>	Returns <i>nil</i> if an error occurs.

Example

For convenience, the character '0' represents white; the character '1' represents black.
Example: to describe an 8x8 pixel icon of a red square surrounded by blue, then thistle on a black background, execute the following (note that spaces are ignored):

Cadence User Interface SKILL Functions Reference

Icons

```
myColorArray = hiCreateColorArray( )
myColorArray[0] = hiMatchColorByName( "thistle")
    ; 'a' denotes thistle

myColorArray[12] = hiMatchColor( list( 1000 0 0 ))
    ; 'm' denotes red

myColorArray[4] = hiMatchColor( list( 0 0 1000))
    ; 'e' denotes blue

iconString = "    1    1    1    1    1    1    1    1\
1  a    a    a    a    a    a    1\
1  a    e    e    e    e    a    1\
1  a    e    m    m    e    a    1\
1  a    e    m    m    e    a    1\
1  a    e    e    e    e    a    1\
1  a    a    a    a    a    a    1\
1  1    1    1    1    1    1    1"

myIcon = hiStringToIcon (myColorArray iconString 8 8)
```

Forms

This chapter describes the following:

- [Creating and Displaying Forms](#) on page 208
- [Standard Forms](#) on page 208
- [Options Forms](#) on page 209
- [Accessing Form Data](#) on page 209
- [Creating Form Fields](#) on page 210
- [Form Layout](#) on page 211
- [Property List Editor](#) on page 218
- [Form and Field Functions](#) on page 219
- [Programming Samples](#) on page 407

Creating and Displaying Forms

You should create a form from the bottom up in the following order:

1. Create all the fields that you need for a form, using the functions for [creating fields](#).
2. Create the form using the [hiCreateAppForm](#) function.
3. Use the [hiDisplayForm](#) function to display the form, or put the form into a window using the [hiOpenWindow](#) function.

Note: Do not use the window manager to close forms or dialog boxes. Close or cancel forms with form buttons such as *Cancel*, *Done*, *Close*, or *No*.

Standard Forms

Standard forms are used to solicit data from the user under normal operation. Each standard form is created with a banner containing the name of the form and five selectable buttons. The buttons perform these functions:

- *OK* applies all changes indicated in the form, executes any user-defined SKILL expression, and removes the form from the screen.
- *Cancel* cancels any changes or selections made to the form, executes any user-defined SKILL expression, and removes the form from the screen.
- *Defaults* sets all values in the form to their default values.
- *Apply* applies the changes indicated in the form and leaves the form displayed for further changes.
- *Help* displays a FrameMaker document containing additional information.

The user works with the form by using the mouse, the keyboard, or a combination of both. When the form first appears, the first type-in field, if any, is active by default. Users can make another field active by selecting it with the mouse, calling `hiSetCurrentField()` specifying a type-in field, or by pressing the `Tab` key to move the cursor to the next text field that accepts focus. If you use the `Tab` key, the cursor must be in the form. Repeatedly hitting the `Tab` key cycles through the text fields that accept focus. When the bottom of the form is reached, the selection continues with the topmost field that can accept focus. Tabbing to a field not currently visible will bring it into view.

Note: For more information about type-in fields that can accept focus through tabbing or clicking the mouse, see the `?fieldFocus` argument of [hiCreateAppForm](#).

Options Forms

Options forms are used only when entering graphical data. Use an options form with `enterFunctions` only. The form is created with a standard banner containing the name of the form and six selectable button boxes. The buttons perform these functions:

- *Hide* applies all changes indicated by the form and then removes the form from the screen.
- *Cancel* calls `cancelEnterFun` and removes the form from the screen.
- *Defaults* sets all values in the form to their default values.
- *Last* cancels any changes or selections made to the form. The selections are returned to the state that existed before the form was brought up.
- *Help* displays a FrameMaker document containing additional information.

Note: Do not use the following functions for options forms. They apply only to standard forms:

`hiSetCallbackStatus`

`hiDisplayForm`

`hiChangeFormCallback`

`hiFormDone`

`hiFormClose`

`hiFormCancel`

`hiFormUnmap`

`hiInstantiateForm`

Accessing Form Data

To access form and field data through SKILL, the “`->`” operator is supported. This may be used to access (read only) the following data (where applicable) from:

```
forms:      hiFormSym
fieldList
_formType
_doneAction
font
dialogStyle
form fields:  hiFieldSym
prompt
value
lastValue
defValue
fieldType
choices
modified
```

Cadence User Interface SKILL Functions Reference

Forms

Note: Do not create fields with any of these symbol names, as this will cause a conflict in the form creation.

The following field data may be set through *formHandle*->*fieldHandle*.

```
value  
defValue
```

For example:

```
hiFormHandle->hiFieldHandle->value = newValue
```

The new settings are immediately reflected in the appearance of the form.

Note: In the above examples, the *formHandle* and *fieldHandle* symbols reference data in the form. They are the SKILL symbols that were specified as arguments to *hiCreateAppForm*() and *hiCreate...Field*.

Creating Form Fields

If all arguments are valid, the internal data structure for a field is returned; otherwise *nil* is returned. The following routines create fields for a form. Each of these routines returns a field descriptor which should be assigned to a user-specified symbol. These symbols will then be used as arguments to the *hiCreateAppForm* routine.

Note: The symbol each field descriptor is *assigned to* may be any arbitrary global symbol. Once these have been passed to the *hiCreateAppForm* function, it is no longer reliable to reference field data through these global symbols. You should then use the *formSymbol*->*fieldSymbol* notation, where *fieldSymbol* is the symbol passed as the ?name argument to the field creation routines.

The following form field creation routines are available:

■ Type-in fields:

```
hiCreateStringField  
hiCreateIntField  
hiCreateFloatField  
hiCreateListField  
hiCreatePointField  
hiCreateBBoxField  
hiCreatePointListField
```

■ Enumerated choice fields:

```
hiCreateToggleField  
hiCreateRadioField
```

Cadence User Interface SKILL Functions Reference

Forms

[hiCreateCyclicField](#)

[hiCreateButtonBoxField](#)

[hiCreateSpinBox](#)

■ Combination fields:

[hiCreateComboField](#)

■ Miscellaneous fields:

[hiCreateButton](#)

[hiCreateLabel](#)

[hiCreateLayerCyclicField](#)

[hiCreateBooleanButton](#)

■ Scroll region fields:

[hiCreateScrollRegion](#)

■ Report fields:

[hiCreateReportField](#)

Form Layout

Form layout is determined by the presence of optional two-dimensional (2D) attributes when the fields of the form are specified. A form or menu is “2D” if attributes are specified for each field. If you specify 2D attributes for one field, you must specify them for all fields. If these attributes are not specified, the form is laid out in a one-dimensional column. Refer to the following sections for a discussion of laying out two-dimensional forms.

One-Dimensional Form Example

The following functions create and display a one-dimensional, standard form:

```
someInt = hiCreateIntField(  
  ?name 'numScoops  
  ?prompt "Number of scoops (0..5)"  
  ?value 2  
  ?defValue 1  
  ?range '(0 5) )  
  
myCyclic = hiCreateCyclicField(  
  ?name 'flavor  
  ?prompt "Flavor"  
  ?value "chocolate"  
  ?choices list("chocolate"  
    "strawberry" "vanilla"  
    "peppermint" "coffee"  
    "green_tea") )  
  
mytoggle = hiCreateToggleField(  
  ?name 'toppings
```

Cadence User Interface SKILL Functions Reference

Forms

```
?prompt "Toppings"
?choices list('(wCream "Whipped
cream?")
list('nuts "any nuts?")
'(jimmies))
?value '(t t nil)
?numSelect 3 )

aRadio = hiCreateRadioField(
?name 'coneSize
?prompt "Cone Size"
?value "Large"
?defValue "Small"

?choices list("Small" "Medium"
"Large" ))

hiCreateAppForm( ?name 'iceCreamForm
?formTitle "Ice Cream"
?callback "buildIceCreamCone()"
?fields list( someInt myCyclic mytoggle aRadio )
?help "cream" )

status = hiDisplayForm( iceCreamForm )
```

The user can change the values of this form through the Command Interpreter Window. The user has a handle into the form and each of its fields through what is created by hiCreateAppForm() (preferably) or hiCreateForm(). For example, the user can change the number of scoops from 2 to 5 by typing

```
iceCreamForm->numScoops->value = 5
```

The cone size can be altered to medium by typing

```
iceCreamForm->coneSize->value = "medium"
```

Two Dimensional Form Layout

The two-dimensional layout of form fields may be obtained by specifying 2D attributes. The specification of these attributes is optional. A form or menu is “2D” if attributes are specified for each field description when creating a form or two-dimensional menu. They are not specified when creating fields themselves. If you specify 2D attributes for one field, you must specify them for all fields. If 2D attributes are not specified, form fields are sequentially laid out in a one-dimensional column.

The following SKILL functions describe the specification of 2D attributes when creating forms and two-dimensional menus. Menus described in this section refer to 2D menus only.

Guidelines

Through two-dimensional attributes, you are given the flexibility of exact placement of fields. No geometry management is provided, and the fields will appear as specified. 2D fields may

Cadence User Interface SKILL Functions Reference

Forms

therefore be specified in any order. Since field coordinates are implicitly obeyed, it is possible to place fields on top of others. The user should keep this in mind, and take care when specifying field locations.

Two Dimensional Attributes

2D attributes are specified in pixel values. The attributes for each field may vary, depending upon how much flexibility you are given. The minimal set of required attributes includes *x*, *y*, *width*, and *height*. The attributes *x* and *y* are coordinates of the upper-left corner of the field, relative to the upper-left (0,0) corner of the form (below the form banner) or menu. The attributes *width* and *height* specify the width and height of the field. These attributes are described in more detail below.

2D attributes are specified at the time the form or menu is created. The attributes should be appended to a list, where the first item of the list is the form field. For example, a 2D field attribute is specified in the following list:

```
list( r_fieldDescription
      x_pos : x_ypos
      x_width : x_height
      [x_promptBoxWidth] )
```

r_fieldDescription

Field description returned from one of the field creation routines.

x_xpos

Upper-left x coordinate in pixels of the field relative to the upper-left corner of the form.

x_ypos

Upper-left y coordinate in pixels of the field relative to the upper-left corner of the form.

x_width

Width of the field in pixels.

x_height

Height of the field in pixels.

x_promptBoxWidth

Width of the prompt box in pixels; only use if prompt is not "". If the prompt is "", no prompt box is created. If a prompt is specified, this attribute is required. The value of the prompt box width should be **less** than the total width of the field (specified by *x_width*). This attribute should not be used for those fields that have no prompts, such as `hiCreateLabel` and `hiCreateButton`.

Cadence User Interface SKILL Functions Reference

Forms

Note: This attribute is *optional* for fields created with `hiCreateBooleanButton`. If not specified, the right edge of the toggle button will appear at the right edge of the field — `x_xpos + x_width`.

For certain 2D attributes, you may specify a constant which takes on a value determined by the size of the form. For this purpose, `hicFormHeight`, `hicFormWidth`, and `hicLineHeight` are available.

<code>hicFormHeight</code>	Used to indicate that the height of this field should span the height of the form. This value is computed at the time the form is created.
<code>hicFormWidth</code>	Used to indicate that the width of this field should span the width of the form. This value is computed at the time the form is created.
<code>hicLineHeight</code>	Used to indicate a default line height value.

Field Attachments

Use field attachments if you want the fields in a two-dimensional form or scroll region to be automatically resized when the form is resized. One-dimensional forms do not support field attachments.

Field attachments are used in conjunction with two-dimensional attributes; they determine how the two-dimensional attributes of a field are used to position that field in the form. Like two-dimensional attributes, field attachments need to be specified when you create a form.

If you want to use field attachments, you must do the following when you create a form:

- Specify the initial size of the form by providing a list of width and height for the `g_initialSize` argument.
- Specify two-dimensional attributes for each field entry in the `l_fieldEntries` argument.
- Provide field attachments for each field listed in `l_fieldEntries`.

You specify field attachments by using the `?attachmentList` `l_fieldAttachments` argument in the function `hiCreateAppForm`. This argument takes a list of attachments, which are applied to the corresponding fields in the `l_fieldEntries` argument. Each attachment is a bitwise OR of a list of constants. Each constant is the attachment specification for a side of the field—left, right, top, or bottom.

Cadence User Interface SKILL Functions Reference

Forms

You can specify either positional attachments or percentage attachments for the sides of a field.

Positional Attachments

If you specify a positional attachment for a side, the side is placed at a fixed offset from the edge of the form. When the form is resized, the width and height of the field increases to accommodate the fixed distance.

Use the following constants to specify positional attachments for the sides of a field:

- `hicLeftPositionSet` (for the left side of the field)
- `hicRightPositionSet` (for the right side of the field)
- `hicTopPositionSet` (for the top side of the field)
- `hicBottomPositionSet` (for the bottom side of the field)

Positional attachments are applied to the two-dimensional attributes of a field in the following way:

If the field attachment is ...	The field's ...
<code>hicLeftPositionSet</code>	Left edge is placed at x .
<code>hicRightPositionSet</code>	Right edge is placed at $x + width [+promptBoxWidth]$
<code>hicTopPositionSet</code>	Top edge is placed at y .
<code>hicBottomPositionSet</code>	Bottom edge is placed at $y + height$.

where x , y , $width$, $height$, and $promptBoxWidth$ are obtained from the two-dimensional attributes of the field, which are specified in the `l_fieldEntries` argument of `hiCreateAppForm`.

If you do not specify an attachment for a side, that side will remain at a fixed distance from the opposite side—at a distance of $width$ pixels if it is the left or right side, or $height$ pixels if it is the top or bottom side. If the opposite side resizes when the form is resized, the non-attached side will move with it to retain the fixed distance.

Note: If you specify both positional and percentage attachments for a side, the positional attachment takes precedence.

Cadence User Interface SKILL Functions Reference

Forms

Percentage Attachments

If you specify a percentage attachment for a side, the side is placed at a percentage of the form width or height. When the form is resized, the width and height will grow to accommodate this fixed percentage. The form width and height are obtained from the *g_initialSize* argument of the function `hiCreateAppForm`.

Use the following constants to specify percentage attachments for the sides of field:

- `hicLeftPercentSet` (for the left side of the field)
- `hicRightPercentSet` (for the right side of the field)
- `hicTopPercentSet` (for the top side of the field)
- `hicBottomPercentSet` (for the bottom side of the field)

Percentage attachments are applied to the two-dimensional attributes of a field in the following way:

If the field attachment is ...	The field's ...
<code>hicLeftPercentSet</code>	Left edge is placed at $x / \text{formWidth} \%$ of <i>formWidth</i> .
<code>hicRightPercentSet</code>	Right edge is placed at $(x + \text{width} + \text{promptBoxWidth}) / \text{formWidth} \%$ of <i>formWidth</i> .
<code>hicTopPercentSet</code>	Top edge is placed at $y / \text{formHeight} \%$ of <i>formHeight</i> .
<code>hicBottomPercentSet</code>	Bottom edge is placed at $(y + \text{height}) / \text{formHeight} \%$ of <i>formHeight</i> .

where *x*, *y*, *width*, *height*, and *promptBoxWidth* are obtained from the two-dimensional attributes of the field (specified in the *l_fieldEntries* argument of `hiCreateAppForm`), and *formWidth* and *formHeight* are obtained from the *g_initialSize* argument of `hiCreateAppForm`.

If you do not specify an attachment for a side, that side will remain at a fixed distance from the opposite side—at a distance of *width* pixels if it is the left or right side, or *height* pixels if it is the top or bottom side. If the opposite side resizes when the form is resized, the non-attached side will move with it to retain the fixed distance.

Note: If you specify both positional and percentage attachments for a side, the positional attachment takes precedence.

Cadence User Interface SKILL Functions Reference

Forms

Example of a Field Attachment

If you want the left and top edges of a field to be at a fixed distance from the upper left corner of the form, but the width and height to grow as a percentage of the size of the form, use the following field attachments for the field:

`hicTopPositionSet | hicLeftPositionSet | hicRightPercentSet | hicBottomPercentSet`

Programming Examples

The following routine creates a one-dimensional form containing a button field and a cyclic field:

```
hiCreateAppForm( ?name 'myForm
  ?formTitle "formName"
  ?fields list( buttonField cyclicField )
  ?callback "doneCallback()"
  ?help "formHelp" )
```

The following routine creates a two-dimensional form containing a field and a cyclic field:

```
hiCreateAppForm( ?name 'myForm
  ?formTitle "formName"
  ?fields list(
    list( buttonField x:y width:height )
    list( cyclicField x2:y2 width2:height2
          promptWidth ) )
  ?callback "doneCallback()"
  ?help "formHelp" )
```

The following routine creates a two-dimensional form with field attachments:

```
hiCreateAppForm( ?name 'myForm
  ?formTitle "formName"
  ?fields list(
    list( buttonField x:y width:height )
    list( cyclicField x2:y2 width2:height2 promptWidth )
  )
  ?attachmentList list( hicTopPositionSet|hicLeftPositionSet|
    hicRightPercentSet|hicBottomPercentSet nil)
  ?initialSize list( width height)
  ?callback "doneCallback()"
  ?help "formHelp" )
```

The following routine creates a 2D menu:

```
hiCreate2DMenu( 'myMenu
  "menuName"
  list( list( buttonField x:y width:height )
    list( labelField x2:y2 width2:height2 ) )
  "menuHelp" )
```

Note: In the above examples, *buttonField* is the value returned from [hiCreateButton](#), *cyclicField* is the value returned from [hiCreateCyclicField](#), and *labelField* is the value returned from [hiCreateLabel](#). The creation of these fields is shown below.

Cadence User Interface SKILL Functions Reference

Forms

Specifying fields for 2D menus is the same as that for 2D forms, but only button and label fields created with `hiCreateButton` and `hiCreateLabel` are accepted. 2D menus are displayed in the same way as popup menus, with `hiDisplayMenu`.

```
buttonField = hiCreateButton( ?name 'someButton
                             ?buttonText "More Options"
                             ?callback "printf(\"pressed\")" )
cyclicField = hiCreateCyclicField( ? name 'cyclic
                                   ?prompt "Cycle through:"
                                   ?value "a"
                                   ?choices '( "a" "b" "c" ) )
labelField = hiCreatelabel( ?name 'myLabel
                            ?labelText "I'm a label"
                            ?justification 'center )
```

To query the extent of the button field, call `hiGetFieldInfo` and look at the contents of *extentList*. For example:

```
extentList = hiGetFieldInfo( myForm 'cyclic )
```

where *extentList* would be of the form

```
(( x2 y2 ) ( width2 height2 ) promptWidth )
```

Property List Editor

You can use the function `hiEditPropList()` to bring up the property list editor for a database object or to create a form (equivalent to the `makeForm` function in earlier versions of the software). The function `hiIgnoreProp()` does not display the specified property of a given object type in the Property List Editor form.

Form and Field Functions

All functions associated with forms are described in this section.

hiAddCyclicChoice

```
hiAddCyclicChoice(  
    r_form  
    s_cyclicField  
    g_newChoice  
)  
=> t / nil
```

Description

Adds *g_newChoice* to the end of the list of choices in the cyclic field in the form.

Note: To change the entire selection of a cyclic field, use

```
form->cyclicField->choices = newchoicelist
```

The new choice list is a list of strings and/or icons for this field.

Arguments

<i>r_form</i>	Form returned from hiCreateAppForm or hiCreateForm .
<i>s_cyclicField</i>	Must be a valid cyclic field in the specified form, and should be specified by its symbol handle within the form. If <i>s_cyclicField</i> is a shared field, <i>g_newChoice</i> is added to all occurrences of that field.
<i>g_newChoice</i>	New choice is added to the <i>cyclicField</i> contained within <i>r_form.newChoice</i> must either be text or an icon list. See hiCreateCyclicField for more information.

Value Returned

t	Returns t if the new item is added.
nil	Returns nil and an error message if the new value is not added.

Cadence User Interface SKILL Functions Reference

Forms

hiAddField

```
hiAddField(  
    r_form  
    g_fieldDescription  
    [i_fieldAttachment]  
)  
=> t / nil
```

Description

Adds a field to a form. The form must already exist, and the field must have been created with one of the field creation routines.

Arguments

r_form Form created by a call to [hiCreateAppForm](#) or [hiCreateForm](#).

g_fieldDescription If you are adding a field to a one-dimensional form, use the following:

r_field

The form field is added to the bottom of the form, keeping the form fields in a column.

If you are adding a field to a two-dimensional form, use the following:

```
list( r_field l_XYlocation l_widthHeight  
    [x_promptBoxWidth])
```

promptBoxWidth is required if a prompt has been specified.

For information about two-dimensional forms, see [“Two Dimensional Form Layout”](#) on page 212.

i_fieldAttachment A bitwise OR of a list of constants. The field attachment determines how the two-dimensional attributes of a field are used to position the field in the form. This argument is valid only for two-dimensional forms that already have field attachments. If

Cadence User Interface SKILL Functions Reference

Forms

you do not specify this argument for forms that have field attachments, the value is assumed to be 0.

For a list of constants and details about how to use field attachments, see [“Field Attachments”](#) on page 214.

Value Returned

<code>t</code>	Returns <code>t</code> if the field is added.
<code>nil</code>	Returns <code>nil</code> and an error message if the field is not added.

Cadence User Interface SKILL Functions Reference

Forms

hiAddFields

```
hiAddFields(  
    r_form  
    l_fieldDescriptions  
    [l_fieldAttachments]  
)  
=> t / nil
```

Description

Adds fields to a form. The form must already exist, and the fields must have been created with one of the field creation routines.

Arguments

r_form Form created by a call to [hiCreateAppForm](#) or [hiCreateForm](#).

l_fieldDescriptions

If you are adding fields to a one-dimensional form, use the following:

```
list( r_field1 r_field2 r_field3 ... )
```

The form field is added to the bottom of the form, keeping the fields in a column.

If you are adding fields to a two-dimensional form, use the following:

```
list(  
    list( r_field1 l_XYlocation l_widthHeight [x_promptBoxWidth])  
    list( r_field2 l_XYlocation l_widthHeight [x_promptBoxWidth])  
    list( r_field3 l_XYlocation l_widthHeight [x_promptBoxWidth])  
    list( r_field4 l_XYlocation l_widthHeight [x_promptBoxWidth])  
)
```

l_fieldAttachments

A bitwise OR of a list of constants. The field attachment determines how the two-dimensional attributes of a field are used to position the field in the form. This argument is valid only for two-dimensional forms that already have field attachments. If you do not specify this argument for forms that have field attachments, the value is assumed to be 0.

Cadence User Interface SKILL Functions Reference

Forms

For a list of constants and details about how to use field attachments, see “[Field Attachments](#)” on page 214.

Value Returned

<code>t</code>	Returns <code>t</code> if the field is added.
<code>nil</code>	Returns <code>nil</code> if the field is not added.

hiChangeCyclicChoices

Description

This function has been replaced with:

```
form->cyclicField->choices = newChoiceList
```

Replaces the choices contained by the cyclic field to the list specified by *newChoiceList*. This is a quick way to replace the choices of a cyclic field without having to delete a cyclic field and insert a field with a new list of choices. As such, the *value*, *lastValue* and *defValue* of the field will be set to the first item of the *newChoiceList* list, and even if the form is instantiated, **no** callback will be called, just as if the field were newly created.

hiChangeFormCallback

```
hiChangeFormCallback(  
    r_form  
    g_newCallback  
)  
=> t
```

Description

Changes the `doneAction` and/or the `cancelAction` of the specified form to the SKILL callback procedure defined by *newCallback*. *newCallback* can be a symbol, a string, or a list of two symbols/strings. The first callback will be taken as the `doneAction`. The second callback will be taken as the `cancelAction`. After this routine is called, if the *OK* or *Apply* button of the form is pressed, the `doneAction` is executed, and if the *Cancel* button is pressed, the `cancelAction` is executed.

Arguments

<i>r_form</i>	Form whose action is being changed.
<i>g_newCallback</i>	The <code>doneAction</code> and/or <code>cancelAction</code> to be executed.

Cadence User Interface SKILL Functions Reference

Forms

hiChangeFormTitle

```
hiChangeFormTitle(  
    r_form  
    t_newTitle  
)  
=> t / nil
```

Description

Changes the window manager title of the specified form to *t_newTitle*.

Arguments

<i>r_form</i>	Form whose title is being changed.
<i>t_newTitle</i>	New title to appear in the window manager frame. This frame is placed around each window depending on the window manager being used.

Value Returned

<i>t</i>	Returns <i>t</i> if the window manager title is changed.
<i>nil</i>	Returns <i>nil</i> if the window manager title is not changed.

hiCreateAppForm

```
hiCreateAppForm(  
    ?name s_name  
    ?fields l_fieldEntries  
    [?attachmentList l_fieldAttachments]  
    [?tabOrderIsAddOrder g_tabOrderIsAddOrder]  
    [?formTitle t_formTitle]  
    [?callback g_callback]  
    [?unmapAfterCB g_unmapAfterCB]  
    [?mapCB S_mapCallback]  
    [?formType s_formType]  
    [?dialogStyle s_dialogStyle]  
    [?buttonLayout g_buttonLayout]  
    [?buttonDisabled s_buttonDisabled]  
    [?help g_help]  
    [?initialSize g_initialSize]  
    [?minSize g_minSize]  
    [?maxSize g_maxSize]  
    [?dontBlock g_dontBlock]  
    [?fieldFocus s_fieldFocus]  
)  
=> r_form
```

Description

Returns the SKILL representation of a form with the specified field entries, setting it to the specified name. All standard forms, from simple forms to property lists, are generated using `hiCreateAppForm`, with varying field entries. Use this function rather than `hiCreateForm`, which does not provide as much functionality.

Arguments

s_name Global SKILL symbol used to reference this form.

Note: Don't use the same symbol to reference different forms.

l_fieldEntries List of field descriptors returned from the field creation routines. You must use at least one field entry. There is no upper limit on the number of field entries you specify.

By default, the fields appear in a column on the form, in the order in which they are listed. To manipulate the order and position of form fields, use two-dimensional attributes. If you use two-dimensional attributes, each field entry for

Cadence User Interface SKILL Functions Reference

Forms

l_fieldEntries will be a list containing a field descriptor and the two-dimensional attributes for that field. For example, a field entry could be

```
list (buttonField x:y width:height).
```

For details about two-dimensional attributes, see [“Two Dimensional Form Layout”](#) on page 212.

Note: If you are creating a form that supports field attachments (so that fields are automatically resized when you resize the form), you must specify two-dimensional attributes for each field entry in *l_fieldEntries*.

l_fieldAttachments

List of field attachments. Field attachments determine how the two-dimensional attributes for a field are used to position that field in the form. The first field attachment is applied to the first field in *l_fieldEntries*, the second to the second field, and so forth. You must have the same number of attachments in *l_fieldAttachments* as the number of fields in *l_fieldEntries*. If you do not want to specify a field attachment for a field, use `nil` or `0`.

Each field attachment in *l_fieldAttachments* consists of a bitwise OR of a list of constants. Each constant is a specification for a side of the field (left, right, top, and bottom). These attachment specifications for the sides of a field can be positional or percentage attachments.

For example, an *l_fieldAttachments* list could be

```
(constant|constant|constant|constant constant|constant  
nil constant|constant|constant)
```

For details about field attachments and how to use positional and percentage constants, see [“Field Attachments”](#) on page 214.

Note: If you use the field attachments argument, you must specify the following:

- The *g_initialSize* argument as a list of width and height.
- Two-dimensional attributes for each field entry in the *l_fieldEntries* argument.

Cadence User Interface SKILL Functions Reference

Forms

g_tabOrderIsAddOrder

t or *nil*. Determines the order in which fields are traversed when the **Tab** key is pressed. If you specify *t*, the *nextField* property is automatically set for the fields and the traversal order is the order in which fields are added to the form (that is, the order in which they are listed in the *l_fieldEntries* list). If you specify *nil*, the window manager's default traversal order is used. The default value of *g_tabOrderIsAddOrder* is *nil*.

Note: If the *nextField* property of a field is already set to anything except *nil*, the value of *g_tabOrderIsAddOrder* does not override it.

t_formTitle

Name that appears in the window manager banner of the form.

g_callback

Used to specify the SKILL functions (callbacks) that execute after you select *Apply* or *OK* or *Cancel* in the form.

g_formAction can be in one of two formats:

g_okAction or
'(*g_okAction* *g_cancelAction*)

g_okAction specifies the SKILL function to be executed when the *Apply* or *OK* button is pressed in a form; *g_cancelAction* specifies the SKILL function to be executed when the *Cancel* button is pressed. Both *g_okAction* and *g_cancelAction* can be in one of the following formats:

t_callback or
s_callback

t_callback is the string representation of the SKILL functions to be executed; *s_callback* is the symbol of the SKILL function to be called (the function will be passed the *r_form* returned from *hiCreateAppForm* as its only parameter).

g_unmapAfterCB

If set to *t*, the form is removed (unmapped) from the screen after the user's callback is executed. If set to *nil*, the form is removed before the callback is invoked. The default value is *nil*. If you use *hiSetCallbackStatus* in your callback routine, you must set the value of *g_unmapAfterCB* to *t* or the callback status will be ignored.

Cadence User Interface SKILL Functions Reference

Forms

S_mapCallback Specifies the callback to be invoked when the form is mapped, that is, when `hiDisplayForm` is called for a form that is not currently mapped. Specify either the name (string) or symbol of the map callback function.

Note: The map callback argument does not apply to forms that are displayed in windows.

s_formType Possible choices include 'nonoptions (default), or 'options. Options forms must be modeless.

s_dialogStyle If set to `systemModal`, the form blocks all other activity on the screen; the user must respond to the form by clicking *OK* or *Cancel* before doing anything else. If set to `systemModal`, the user cannot access the *Help* button and/or context-sensitive help before responding to the form. If set to `modal`, the form blocks all other Cadence software activity, but not other activity on the screen. If set to `modeless`, the form does not block any activity.

Default Value: `modeless`

g_buttonLayout Used to specify the buttons that appear in the banner of the form. *g_buttonLayout* can be in one of two formats:

s_buttonLayout or
'(s_buttonLayout (s_buttonText st_buttonCB) ...)

s_buttonLayout specifies one of the accepted base set of button layouts. The *Help* button is automatically added with each of these combinations. The following combinations are accepted:

For nonoptions forms:

'OKCancel
'OKCancelDef
'OKCancelLast
'OKCancelDefLast
'OKCancelApply
'ApplyCancel
'ApplyCancelDef
'ApplyCancelLast
'ApplyCancelDefLast

Cadence User Interface SKILL Functions Reference

Forms

```
'Close  
'OKCancelDefApply (default)
```

For options forms:

```
'HideCancel  
'HideCancelDef  
'HideCancelLast  
'HideCancelDefLast (default)  
'ApplyHideCancel  
'ApplyHideCancelDef  
'ApplyHideCancelLast  
'ApplyHideCancelDefLast
```

where

OK denotes the *OK* button

Apply denotes the *Apply* button

Cancel denotes the *Cancel* button

Close denotes the *Close* button (the *Close* button is similar to the *OK* button)

Last denotes the *Last* button

Def denotes the *Defaults* button

Hide denotes the *Hide* button

For user-defined functions, following the *s_buttonLayout* argument should be lists consisting of buttonText/buttonCallback pairs:

```
'(s_buttonText st_buttonCB)
```

s_buttonText should be the symbol whose printname is the text to appear in the button. For example, to have a button that has the text: "Press Me", *s_buttonText* would be specified as 'Press\ Me. The callback for this button can be specified as a symbol or string. If it is a symbol, the function will be passed the *r_form* returned from *hiCreateAppForm* as its only parameter. If it is a string, the unchanged string will be passed to SKILL to interpret.

Previous and *Next* are special case buttons. If specified, their buttonText/buttonCallback lists should appear as a pair, follow the base button symbol, and precede all other user defined buttons. For example:

Cadence User Interface SKILL Functions Reference

Forms

```
'(OKCancel
  (Previous "myPreviousCB()")
  (Next "myNextCB()")
  (Some\ other\ button otherCB))
```

would be an acceptable way to specify a *Previous* and *Next* button, followed by a button with "Some other button" as its text.

Note: The action of carriage return, <CR>, performs the same action as the first button of the form button list, which is usually `hiFormDone()` for nonoptions forms, and `hiToggleEnterForm()` for options forms. If you specify an alternate button list, however, the action of <CR> may change. For example, if your button list is `'ApplyCancelDef`, the action of <CR> in a type-in field performs the `hiFormApply()` action.

s_buttonDisabled

List of buttons that are disabled when the form appears. You can change this value with the [hiSetFormButtonEnabled](#) function.

g_help

Specify *g_help* as a single string, single symbol, list of strings, or list of symbols that will be used to reference a .tgf help file.

If you specify a single string or symbol, it is used to set the `hiHelpAppName` property on the symbol that represents the form name. If you specify a list of strings or a list of symbols, the first element in the list is used to set the `hiHelpAppName` property and the second element is used to set the `hiHelpSymName` property.

When a user clicks *Help* on the form, the `hiHelp` function is called with the following arguments:

appId helpTag

If the `hiHelpAppName` property has been set, it is used as the *appId*, otherwise the application ID of the current window is used as the *appId*. If the `hiHelpSymName` property has been set, it is used as the *helpTag*, otherwise the form name is used as the *helpTag*.

Cadence User Interface SKILL Functions Reference

Forms

The *appId* and *helpTag* determine the name of the tag file that is called (*appId.tgf*) and the help tag in the file that is accessed.

g_initialSize The size of the form when it first appears. *g_initialSize* accepts the following values:

- A list of width and height. The form is sized to the specified width and height or to the size needed to accommodate all the fields, whichever is less.
- A symbol representing a field in the form. The form is sized to include the field but exclude any other fields to the right of and below the specified field.
- *t*. The form is sized to fit all fields (or to the size of the screen, whichever is smaller).

Note: If you are creating a form that supports field attachments (so that fields are automatically resized when you resize the form), you must specify the *g_initialSize* argument and it must be a list of width and height.

g_minSize The minimum size of the form, specified as a list of width and height or *width:height*. The minimum size excludes the window manager's title and border. The default minimum size is the value of the *g_initialSize* argument.

To change the minimum size, use the [hiSetFormMinMaxSize](#) function.

g_maxSize The maximum size of the form, specified as a list of width and height or *width:height*. The maximum size excludes the window manager's title and border. The default maximum size is the size of the screen.

To change the maximum size, use the [hiSetFormMinMaxSize](#) function.

g_dontBlock If set to *t*, a call to `hiDisplayForm()` on this form will immediately return *t*. The default is *nil*.

Previous to the 9504 release, `hiDisplayForm` would not return *t* or *nil* immediately. The form used to be displayed until the user would select either the *OK* or the *Cancel* button. If the user would select the *OK* button, it would return *t*. If the form was cancelled, it would return *nil*.

Cadence User Interface SKILL Functions Reference

Forms

s_fieldFocus

Optional argument used to determine which type-in fields can accept focus through tabbing or clicking the mouse. Possible choices are 'editableTypein (default) and 'allTypein.

If set to 'allTypein, all type-in fields can accept focus, whether they are editable or non-editable. If set to 'editableTypein, only editable type-in fields can accept focus.

Value Returned

r_form

SKILL structure representing the form.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateBBoxField

```
hiCreateBBoxField(  
    ?name s_fieldName  
    [prompt t_fieldPrompt]  
    [?value l_currentValue]  
    [?help g_fieldHelp]  
    [?defValue l_defaultValue]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a bounding box field for a form.

A bounding box field accepts only a bounding box as input. A valid bounding box is represented by two points (each point is a list of two integers) separated by blank space. The first point is the lower-left corner of the box, the second point is the upper-right corner.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_fieldPrompt</i>	Optional prompt for this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>l_currentValue</i>	Initial and current value of this field in the correct format. This is a required parameter. May be <code>nil</code> , in which case the <i>bBox</i> field appears blank. To reference the current value, use <code>formHandle->hiFieldSym->value</code>
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.

Cadence User Interface SKILL Functions Reference

Forms

<i>l_defaultValue</i>	<p>Default value of the field (in the correct format). This value is equal to the initial entry in <i>l_currentValue</i> if <i>l_defaultValue</i> is not specified. To reference the default value, use</p> <pre>formHandle->fieldSym->defValue</pre> <pre>formHandle->hiFieldSym->defValue</pre>
<i>t_font</i>	<p>An optional font argument is accepted but not supported at this time.</p>
<i>t_callback</i>	<p>One or more SKILL functions to be executed if a field's value has changed. The callback is called when the field has lost focus (when the cursor leaves the form, the user tabs to another field, and so forth) or when the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the <i>t_modifyCallback</i> argument.</p> <p>Note: If you specify multiple SKILL statements for execution, they must be enclosed in curly braces: {}.</p>
<i>t_modifyCallback</i>	<p>Note: Do not use the <i>t_modifyCallback</i> argument with this field because it may not work as expected. Use this argument only with string fields.</p> <p>SKILL function to be executed whenever any change is made in the text field, for example, by typing or pasting. The modify callback is called when a change is detected in the field but before the change is displayed.</p> <p>The modify callback function is passed the following arguments:</p> <pre>s_fieldName t_latestTextValue g_sourceOfChange</pre> <p>where <i>s_fieldName</i> is the symbol of the field in which the change was made, <i>t_latestTextValue</i> is the latest text value of the field, and <i>g_sourceOfChange</i> is <i>t</i> or <i>nil</i> where <i>t</i> indicates that the change was made programmatically (by changing the value of the field) and <i>nil</i> indicates that the change was made by a user action.</p> <p>The modify callback function should return the following:</p>

Cadence User Interface SKILL Functions Reference

Forms

`t | nil | value`

If the modify callback function returns `t`, the changes made to the field are allowed and are displayed as they are entered. If the function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be a bounding box. This value replaces the current value of the field.

`g_editable`

If set to `nil`, the field is read-only; setting *`g_editable`* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *`?fieldFocus`* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#).

`g_enabled`

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

`g_invisible`

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

`g_nextField`

Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this argument, the Tab traversal order is determined by the *`tabOrderIsAddOrder`* argument of [hiCreateAppForm](#) or [hiCreateScrollRegion](#). If the *`tabOrderIsAddOrder`* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Value Returned

`r_fieldHandle`

Handle to the bounding box field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateBooleanButton

```
hiCreateBooleanButton(  
    ?name s_fieldName  
    ?buttonText t_buttonText  
    [?buttonLocation s_buttonLocation]  
    [?callback t_callback]  
    [?value g_booleanValue]  
    [?defValue g_defaultValue]  
    [?help g_fieldHelp]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a single Boolean button for a form. The button text appears as a prompt next to the button; the button is on the left of the button text if the `?buttonLocation` argument is `'left` and on the right of the button text if the `?buttonLocation` argument is `'right`. A Boolean field consists of a single toggle button which may have an associated callback procedure executed when the Boolean state changes.

For efficiency, Boolean buttons should always be used in preference to single toggle buttons.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_buttonText</i>	Prompt string which appears next to the toggle button.
<i>s_buttonLocation</i>	<code>'left</code> or <code>'right</code> Default value: <code>'right</code> Specify <code>'left</code> if you want the button to appear on the left of the button text; specify <code>'right</code> if you want the button to appear on the right of the button text. If the prompt box width is not specified and the button is on the right, the button is right justified in the field. If the prompt box width is not specified and the button is on the left, the button text

Cadence User Interface SKILL Functions Reference

Forms

on the right is not right justified; instead, it is placed as if the prompt box width were set to 20. (The prompt box width is a two-dimensional attribute you can specify when you create the form.)

Note: In one-dimensional forms, the button text is always on the left; the `?buttonLocation` argument is ignored.

t_callback

One or more SKILL functions to be executed if a field's value has changed. The callback is called as soon as the button state changes.

Note: To specify multiple SKILL statements for execution, they must be enclosed by curly braces: `{}`.

g_booleanValue

Value of Boolean button field (`t` or `nil`). If not specified, it defaults to `nil`.

g_defaultValue

Optional default value of the field. This value is equal to the initial entry in *currentValue* if *defaultValue* is not specified. To reference the default value, use

```
formHandle->hiFieldSym->defValue
```

g_fieldHelp

A string or symbol used to reference help. If not specified, *s_fieldName* is used. This argument is currently not used.

t_font

A font argument is accepted but not supported at this time.

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle

Handle to the Boolean button.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateButton

```
hiCreateButton(  
    ?name s_fieldName  
    ?buttonText t_buttonText  
    ?callback t_callback  
    [?buttonIcon l_buttonIcon]  
    [?help g_fieldHelp]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a standalone push button for a form or two-dimensional menu. The button must have an associated callback procedure to be executed when the button is selected. A button only field differs from a button box field in that there is no prompt associated with this button.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_buttonText</i>	String that appears on the button.
<i>t_callback</i>	One or more SKILL functions to be executed if the button is activated. Note: To specify multiple SKILL statements for execution, they must be enclosed by curly braces: <code>{ }</code> .
<i>l_buttonIcon</i>	An icon list returned from <code>geCellViewToDlist</code> , <code>dlDlistToIcon</code> or <code>hiStringToIcon</code> . Overrides any <i>buttonText</i> specified.
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_font</i>	A font argument is accepted but is not supported at this time.

Cadence User Interface SKILL Functions Reference

Forms

g_enabled *t* or *nil*. Specify *t* if you want the field enabled; specify *nil* if you want the field disabled, that is, greyed-out. The default value is *t*.

g_invisible *t* or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle Handle to the button.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateButtonBoxField

```
hiCreateButtonBoxField(  
    ?name s_fieldName  
    ?choices l_buttonText  
    ?callback l_callbackList  
    ?prompt t_fieldPrompt  
    [?help g_fieldHelp]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a button box field for a form. A button box field contains individual buttons to be displayed in one row. Each button in the button box will have an associated callback procedure that is executed when that button is selected. A prompt is displayed on this row as well.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>l_buttonText</i>	List of text strings representing the text displayed within each individual button in the button box (in left-to-right order).
<i>l_callbackList</i>	List of callback strings. The number of button text entries must equal the length of the <i>l_callbackList</i> . The callback is executed when the corresponding button is pressed.
<i>t_fieldPrompt</i>	Prompt that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_font</i>	A font argument is accepted but is not supported at this time.

Cadence User Interface SKILL Functions Reference

Forms

g_enabled *t* or *nil*. Specify *t* if you want the field enabled; specify *nil* if you want the field disabled, that is, greyed-out. The default value is *t*.

g_invisible *t* or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle Handle to the button box field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateComboField

```
hiCreateComboField(  
    ?name s_fieldName  
    ?items g_choices  
    [?prompt t_fieldPrompt]  
    [?value t_currentValue]  
    [?help g_fieldHelp]  
    [?defValue t_defaultValue]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?format t_fieldFormat]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a drop-down combo field, which is a type-in field with an attached drop-down list. Users can either select an item from the list or type in a value.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>g_choices</i>	A list of strings that represent the choices in the drop-down list. The default value is <code>nil</code> . To reference it, use <code>formHandle->hiFieldSym->items</code>
<i>t_fieldPrompt</i>	Optional prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>t_currentValue</i>	Initial and current value of this field. To reference it, use <code>formHandle->hiFieldSym->value</code>

Cadence User Interface SKILL Functions Reference

Forms

<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_defaultValue</i>	<p>Optional, default value of the field. This value is equal to the initial entry in <i>t_currentValue</i> if <i>t_defaultValue</i> is not specified. To reference the default value, use</p> <pre>formHandle->hiFieldSym->defValue</pre>
<i>t_font</i>	An optional font argument is accepted but is not supported at this time.
<i>t_callback</i>	<p>One or more SKILL functions to be executed if a field's value has changed. The callback is called when the field has "lost focus" (for example, when the the user moves the cursor outside the form or tabs to another field) or when the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the <i>t_modifyCallback</i> argument.</p> <p>Note: If you specify multiple SKILL statements for execution, they must be enclosed by curly braces: {}.</p>
<i>t_modifyCallback</i>	<p>SKILL function to be executed whenever any change is made in the text field, for example, by typing a value. The modify callback is called when a change is detected in the field but before the change is displayed.</p> <p>The modify callback function is passed the following arguments:</p> <pre>s_fieldName t_latestTextValue g_sourceOfChange</pre> <p>where <i>s_fieldName</i> is the symbol of the field in which the change was made, <i>t_latestTextValue</i> is the latest text value of the field, and <i>g_sourceOfChange</i> is <i>t</i> or <i>nil</i> where <i>t</i> indicates that the change was made programmatically (by changing the value of the field) and <i>nil</i> indicates that the change was made by a user action.</p> <p>The modify callback function should return the following:</p> <pre>t nil value</pre>

Cadence User Interface SKILL Functions Reference

Forms

If the modify callback function returns `t`, the changes made to the field are allowed and are displayed as they are entered. If the function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be a string. This value replaces the current value of the field.

t_fieldFormat

Optional string that can control the display of the value on the form. It is similar to `printf` and its default value is `"%s"`.

Note: Acceptable format characters are those allowed by the SKILL programming language.

g_editable

If set to `nil`, the field is read-only; setting *g_editable* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#).

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_nextField

Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this argument, the Tab traversal order is determined by the *tabOrderIsAddOrder* argument of `hiCreateAppForm` or `hiCreateScrollRegion`. If the *tabOrderIsAddOrder* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Value Returned

r_fieldHandle

Handle to the drop-down combo field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateCyclicField

```
hiCreateCyclicField(  
    ?name s_fieldName  
    ?choices l_enumerations  
    ?prompt t_fieldPrompt  
    [?value g_currentValue]  
    [?defValue g_defaultValue]  
    [?callback t_callback]  
    [?help g_fieldHelp]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a cyclic field for a form. A cyclic field contains a list of string or iconic enumerations. Only one item may be chosen at a time. Using the mouse, the user displays all cyclic choices in pulldown menu form. Once the user has brought up this cyclic choice menu, a selection may be made from it.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>l_enumerations</i>	List of strings and/or icons that can be cycled through. The current value and default value (if any) must be contained within this enumerated list.
<i>t_fieldPrompt</i>	Prompt string that appears in this field (it may be optional for some fields). To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>g_currentValue</i>	The current value of the cyclic field. Must be a string or icon list specified in <i>l_enumerations</i> . If <i>g_currentValue</i> is not specified, it defaults to the first choice in the enumerations list. To reference the current value, use <code>formHandle->hiFieldSym->value</code>

Cadence User Interface SKILL Functions Reference

Forms

<i>g_defaultValue</i>	<p>Default value of the field (in the correct format). This value is equal to the initial entry in <i>g_currentValue</i> if <i>g_defaultValue</i> is not specified. To reference the default value, use</p> <pre>formHandle->hiFieldSym->defValue</pre>
<i>t_callback</i>	<p>Callback string for this field. If a callback is specified, the item selected (either text or an icon) is appended to this callback before being executed by SKILL.</p> <p>An iconic value in the enumerations list must be a list returned from an icon-generating function (<i>geCellViewToDlist</i>, <i>dlDlistToIcon</i> or <i>hiStringToIcon</i>). If the user selects an icon value in this field, then</p> <pre>form->field->value</pre> <p>is set to the icon list. You may associate a value or values with these icons by creating a multi-element list with <i>list</i>. The first element of this list must be the icon list. The second element may be user-defined data. When the value is set to an icon, you can retrieve the user data from this list. For example:</p> <pre>layer1Icon = dlDlistToIcon(...) myIcon = dlDlistToIcon(...) layer1List = list(layer1Icon getLayerId("layer1")) myIconList = list(myIcon list("myIcon" 33)) field = hiCreateCyclicField(?name 'icons ?prompt "Choose one:" ?choices list(layer1List myIconList "red" "yellow") ?value myIconList)</pre> <p>In the above example,</p> <pre>form->field->value</pre> <p>contains something like:</p> <pre>((345233 10 10) ("myIcon" 33))</pre> <p>where the first element describes the icon.</p>
<i>g_fieldHelp</i>	<p>A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.</p>

Cadence User Interface SKILL Functions Reference

Forms

<i>t_font</i>	A font argument is accepted but is not supported at this time.
<i>g_enabled</i>	<code>t</code> or <code>nil</code> . Specify <code>t</code> if you want the field enabled; specify <code>nil</code> if you want the field disabled, that is, greyed-out. The default value is <code>t</code> .
<i>g_invisible</i>	<code>t</code> or <code>nil</code> . Specify <code>t</code> if you want the field to be invisible; specify <code>nil</code> if you want the field to be visible. The default value is <code>nil</code> .

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

<i>r_fieldHandle</i>	Handle to the cyclic field.
----------------------	-----------------------------

Cadence User Interface SKILL Functions Reference

Forms

hiCreateFloatField

```
hiCreateFloatField(  
    ?name s_fieldName  
    [?value f_currentValue]  
    [?acceptNil g_acceptNil]  
    [?prompt t_fieldPrompt]  
    [?defValue f_defaultValue]  
    [?help g_fieldHelp]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?range l_fieldRange]  
    [?format t_fieldFormat]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a float field for a form. A float field is a field which accepts floating-point input only. A format string can be specified which will automatically be imposed on any user input. An optional range can be specified and enforced for this field as well.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>f_currentValue</i>	Initial and current value of this field. It is a required parameter, and may be referenced as <code>formHandle->hiFieldSym->value</code>
<i>g_acceptNil</i>	If set to <code>t</code> , the field will accept <code>nil</code> as a value. If there is no value entered for the field on the form, it will result in the SKILL value being set to <code>nil</code> . Setting a field value to <code>nil</code> using the CIW will result in the field on the form to be blank.
<i>t_fieldPrompt</i>	Optional prompt string for this field. To reference it, use

Cadence User Interface SKILL Functions Reference

Forms

	<code>formHandle->hiFieldSym->prompt</code>
<code>f_defaultValue</code>	Optional, default value of the field (in the correct format). This value is equal to the initial entry in <code>currentValue</code> if <code>defaultValue</code> is not specified. To reference the default value, use <code>formHandle->hiFieldSym->defValue</code>
<code>g_fieldHelp</code>	A string or symbol used to reference help. If not specified, <code>s_fieldName</code> is used. This argument is currently not used.
<code>t_font</code>	A font argument is accepted but not supported at this time.
<code>t_callback</code>	One or more SKILL functions to be executed if a field's value has changed. The callback is called when the field has "lost focus" (when the cursor leaves the form, the user tabs to another field, and so forth) or when the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the <code>t_modifyCallback</code> argument. Note: If you specify multiple SKILL statements for execution, they must be enclosed in curly braces: {}.
<code>t_modifyCallback</code>	Note: Do not use the <code>t_modifyCallback</code> argument with this field because it may not work as expected. Use this argument only with string fields. SKILL function to be executed whenever any change is made in the text field, for example, by typing or pasting. The modify callback is called when a change is detected in the field but before the change is displayed. The modify callback function is passed the following arguments: <code>s_fieldName t_latestTextValue g_sourceOfChange</code> where <code>s_fieldName</code> is the symbol of the field in which the change was made, <code>t_latestTextValue</code> is the latest text value of the field, and <code>g_sourceOfChange</code> is <code>t</code> or <code>nil</code> where <code>t</code> indicates that the change was made programmatically (by changing the value of the field) and <code>nil</code> indicates that the change was made by a user action.

Cadence User Interface SKILL Functions Reference

Forms

The modify callback function should return the following:

`t | nil | value`

If the modify callback function returns `t`, the changes made to the field are allowed and are displayed as they are entered. If the function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be a floating point. This value replaces the current value of the field.

l_fieldRange

Optional list containing the minimum and maximum float values. For example, the list `(0.5 10.9)`, also specified as `0.5:10.9`, represents the range of floats from 0.5-10.9 inclusive. To specify an unbounded upper range, use either `nil` or the string "infinity" as the second value. To specify an unbounded lower range, use either `nil` or the string "-infinity" as the first value.

t_fieldFormat

Optional string that can control the display of the value on the form. It is similar to `printf`.

Note: Acceptable format characters are those allowed by the C programming language.

g_editable

If set to `nil`, the field is read-only; setting *g_editable* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to 'allTypein. For more information, see [hiCreateAppForm](#).

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

Cadence User Interface SKILL Functions Reference

Forms

g_nextField Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this argument, the Tab traversal order is determined by the *tabOrderIsAddOrder* argument of `hiCreateAppForm` or `hiCreateScrollRegion`. If the *tabOrderIsAddOrder* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Value Returned

r_fieldHandle Handle to the float field.

hiCreateFrameField

```
hiCreateFrameField(  
    ?name s_fieldName  
    [?labelText t_labelText]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a frame field for a two-dimensional form. A frame field is a border that surrounds a group of related fields so they are easier to see.

When you specify the 2D attributes for a frame field, if *t_labelText* is specified, you must also specify the label height. For example:

```
list( r_frameField x_xpos:x_ypos x_width:x_height x_labelHeight)
```

x_labelHeight must be less than the total *x_height* of the field. The upper left corner of the frame appears at *x_xpos*:*x_ypos*+*x_labelHeight*.

Arguments

<i>s_fieldName</i>	Handle to the frame field.
<i>t_labelText</i>	Title of the frame field. If the title is specified it appears flush left above the upper left corner of the frame. The title must not extend beyond the width of the frame.
<i>g_enabled</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want the field enabled; specify <i>nil</i> if you want the field disabled, that is, greyed-out. The default value is <i>t</i> .
<i>g_invisible</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want the field to be invisible; specify <i>nil</i> if you want the field to be visible. The default value is <i>nil</i> .

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

r_fieldHandle Field handle for the frame field.

hiCreateScrollRegion

```
hiCreateScrollRegion(  
    ?name s_name  
    [?fields l_fieldEntries]  
    [?attachmentList l_fieldAttachments]  
    [?tabOrderIsAddOrder g_tabOrderIsAddOrder]  
    [?labelText t_labelText]  
    [?borderWidth x_borderWidth]  
    [?scrollBars g_scrollBars]  
    [?rightMargin x_rightMargin]  
    [?bottomMargin x_bottomMargin]  
    [?help g_help]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a scroll region field, which can be scrolled and can contain any type of field. A scroll region field is like a sub-form within a form.

A scroll region field can be used as a field only on 2D forms. (See [“Form Layout”](#) on page 211 for information about 2D forms.) Once the field is instantiated, it is referenced through the form that contains it (see the description for the *s_name* argument below for more information).

A scroll region field can be specified as a form field for all functions that take form field arguments. It can also be specified as the form argument for all functions for adding, moving, resizing and deleting fields from a form.

Note: Do not place a scroll region field twice in one form or two scroll region fields with the same name in one form.

Note: You cannot place a scroll region field that has field attachments inside a form or scroll region field that does not have field attachments.

Arguments

s_name Handle to the scroll region field within a form. Once the form is instantiated, the field is referenced through the form. For example, to update the fields or access properties in a scroll region field *myRegion* which is in a form *myForm*, you use

```
myForm->myRegion
```

Cadence User Interface SKILL Functions Reference

Forms

l_fieldEntries

List of field descriptors returned from the field creation routines. These can be fields of any type, including other scroll region fields. You must use at least one field entry. There is no upper limit on the number of fields you can use. You must specify 2D attributes for each field entry. (See [“Form Layout”](#) on page 211 for information about 2D forms.)

If the field entries list includes any scroll region fields, the scroll region field is copied. Scroll region fields are never shared.

All fields placed in a scroll region field must be referenced through the form and scroll region field. If the scroll region field is within another scroll region field, then its fields are accessed through the form and both scroll region fields. For example, to access the value of a field `myField`, which is in a scroll region field `myRegion` in a form `myForm`, you use

```
myForm->myRegion->myField->value
```

In the above example, if you also have a `myRegion2` with a `myField2`, then to access the value of `myField2` you use

```
myForm->myRegion->myRegion2->myField2->value
```

l_fieldAttachments

List of field attachments. Field attachments determine how the two-dimensional attributes for a field are used to position that field in the scroll region. The first field attachment is applied to the first field in *l_fieldEntries*, the second to the second field, and so forth. You must have the same number of attachments in *l_fieldAttachments* as the number of fields in *l_fieldEntries*. If you do not want to specify a field attachment for a field, use `nil` or `0`.

Each field attachment in *l_fieldAttachments* consists of a bitwise OR of a list of constants. Each constant is a specification for a side of the field (left, right, top, and bottom). These attachment specifications for the sides of a field can be positional or percentage attachments.

For example, an *l_fieldAttachments* list could be

Cadence User Interface SKILL Functions Reference

Forms

(constant | constant | constant | constant constant |
constant nil constant | constant | constant)

For details about field attachments and how to use positional and percentage constants, see [“Field Attachments”](#) on page 214.

Note: You cannot place a scroll region field that has field attachments inside a form or scroll region field that does not have field attachments.

g_tabOrderIsAddOrder

t or *nil*. Determines the order in which fields are traversed when the **Tab** key is pressed. If you specify *t*, the *nextField* property is automatically set for the fields and the traversal order is the order in which fields are added to the form (that is, the order in which they are listed in the *l_fieldEntries* list). If you specify *nil*, the window manager’s default traversal order is used. The default value of *g_tabOrderIsAddOrder* is *nil*.

Note: If the *nextField* property of a field is already set to anything except *nil*, the value of *g_tabOrderIsAddOrder* does not override it.

t_labelText

Title of the scroll region field. It will be justified to the upper left corner of the bounding box of the field.

x_borderWidth

Thickness of the border shadow in pixels. The default is 2. When scrollbars appear, the border will shrink in size by the amount of space taken up by the scrollbars. The right or bottom edge of the scrollbar will be where the right or bottom edge of the border was.

This property can be changed interactively at any time, even after the scroll region field has been instantiated.

g_scrollBars

Controls the appearance of scrollbars. Can have the following values:

'dynamic

Scrollbars automatically appear if a field that extends beyond the edge of the scroll region field is added.

Cadence User Interface SKILL Functions Reference

Forms

`'static/t` Scrollbars are always present, whether or not a field extends beyond the boundary of the scroll region field.

`nil` Scrollbars are never present, even if a field extends beyond the boundary of the scroll region field.

`(t|nil t|nil)` Controls each scrollbar—horizontal and vertical—individually. If the first element in the list is `t`, the horizontal scrollbar will always be present; if it is `nil`, the horizontal scrollbar will never be present. If the second element in the list is `t`, the vertical scrollbar will always be present; if it is `nil`, the vertical scrollbar will never be present.

`'dynamic` or `'static` cannot be used as values in this list.

Note: If you use this list as the value of `g_scrollBars`, the horizontal and vertical scrollbars cannot be made dynamic.

Note: The scrollbars property can only be set when you create the scroll region field. Once the field is instantiated, you cannot change this property.

`x_rightMargin`

Distance, in pixels, between the right edge of the border shadow and the edge of the field. The default is 0.

The visible part of the scroll region is the width of the field minus the `x_rightMargin`.

This property can be changed interactively at any time, even after the scroll region field is instantiated.

`x_bottomMargin`

Distance, in pixels, between the bottom edge of the border shadow and the edge of the field. The default is 0.

The visible part of the scroll region is the height of the field minus the `x_bottomMargin`.

Cadence User Interface SKILL Functions Reference

Forms

This property can be changed interactively at any time, even after the scroll region field is instantiated.

g_help

A string, symbol, list of strings, or list of symbols used to reference help. If not specified, *s_name* is used. This argument is currently not used.

g_invisible

t or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle

Handle to the scroll region field created.

hiIsScrollRegion

```
hiIsScrollRegion(  
    g_item  
)  
=> t/nil
```

Description

Returns *t* if the item is a scroll region field, *nil* if it is not a scroll region field.

Arguments

<i>g_item</i>	Item to be checked.
---------------	---------------------

Value Returned

<i>t</i>	If the function returns <i>t</i> , the item is a scroll region field.
----------	---

<i>nil</i>	If the function returns <i>nil</i> , the item is not a scroll region field.
------------	---

hiCreateForm

```
hiCreateForm(  
    s_formHandle  
    t_formName  
    g_formAction  
    l_fieldEntries  
    t_formHelp  
    [g_unmapAfterCB]  
)  
=> r_form
```

Description

hiCreateForm is now just a wrapper function to hiCreateAppForm. Forms, from simple forms to property lists, are generated using hiCreate{App}Form, with varying field entries.

You should no longer use hiCreateForm since new functionality is only added to hiCreateAppForm.

Use hiCreateAppForm instead of this function.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateFormButton

```
hiCreateFormButton(  
    ?name s_fieldName  
    ?buttonText t_buttonText  
    ?callback t_callback  
    [?buttonIcon l_buttonIcon]  
    [?help g_fieldHelp]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a standalone push button for a form or two-dimensional menu. Same as `hiCreateButton`.

The button must have an associated callback procedure to be executed when the button is selected. A button only field differs from a button box field in that there is no prompt associated with this button.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_buttonText</i>	String that appears on the button.
<i>t_callback</i>	One or more SKILL functions to be executed if the button is activated. Note: To specify multiple SKILL statements for execution, they must be enclosed by curly braces: <code>{ }</code> .
<i>l_buttonIcon</i>	Overrides any <i>buttonText</i> specified. An icon list returned from <code>geCellViewToDlist</code> , <code>dldlistToIcon</code> or <code>hiStringToIcon</code> .
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.

Cadence User Interface SKILL Functions Reference

Forms

<i>t_font</i>	A font argument is accepted but is not supported at this time.
<i>g_enabled</i>	<code>t</code> or <code>nil</code> . Specify <code>t</code> if you want the field enabled; specify <code>nil</code> if you want the field disabled, that is, greyed-out. The default value is <code>t</code> .
<i>g_invisible</i>	<code>t</code> or <code>nil</code> . Specify <code>t</code> if you want the field to be invisible; specify <code>nil</code> if you want the field to be visible. The default value is <code>nil</code> . You can modify the value of this argument (after the form is created) with the <code>invisible</code> property. For example: <code>formHandle->fieldSym->invisible = nil</code>

Value Returned

<i>r_fieldHandle</i>	Handle to the button.
----------------------	-----------------------

Reference

[hiCreateButton](#), [hiCreateFormButton](#), [hiDeleteForm](#)

Cadence User Interface SKILL Functions Reference

Forms

hiCreateFormLabel

`hiCreateFormLabel()`

Description

This function is an alias for `hiCreateLabel`. Use `hiCreateLabel` instead of this function.

Reference

[hiCreateLabel](#)

Cadence User Interface SKILL Functions Reference

Forms

hiCreateIntField

```
hiCreateIntField(  
    ?name s_fieldName  
    [?prompt t_fieldPrompt]  
    [?value x_currentValue]  
    [?acceptNil g_acceptNil]  
    [?help g_fieldHelp]  
    [?defValue x_defaultValue]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?format t_fieldFormat]  
    [?range l_fieldRange]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates an integer field entry for a form. An `int` field is a field that accepts integer input only. A format string can be specified which will automatically be imposed on any user input. An optional range can be specified and enforced for this field as well.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_fieldPrompt</i>	Prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>x_currentValue</i>	Initial and current value of this field. To reference it, use <code>formHandle->hiFieldSym->value</code>
<i>g_acceptNil</i>	If set to <code>t</code> , the field will accept <code>nil</code> as a value. If there is no value entered for the field on the form, it will result in the SKILL value being set to <code>nil</code> . Setting a field value to <code>nil</code> using the CIW will result in the field on the form to be blank.

Cadence User Interface SKILL Functions Reference

Forms

<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>x_defaultValue</i>	<p>Default value of the field. This value is equal to the initial entry in <i>currentValue</i> if <i>defaultValue</i> is not specified. To reference the default value, use</p> <pre>formHandle->hiFieldSym->defValue</pre>
<i>t_font</i>	A font argument is accepted but not supported at this time.
<i>t_callback</i>	<p>One or more SKILL functions to be executed if a field's value has changed. The callback is called when the field has lost focus (when the cursor leaves the form, the user tabs to another field, and so forth) or the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the <i>t_modifyCallback</i> argument.</p> <p>Note: If you specify multiple SKILL statements for execution, they must be enclosed in curly braces: {}.</p>
<i>t_modifyCallback</i>	<p>Note: Do not use the <i>t_modifyCallback</i> argument with this field because it may not work as expected. Use this argument only with string fields.</p> <p>SKILL function to be executed whenever any change is made in the text field, for example, by typing or pasting. The modify callback is called when a change is detected in the field but before the change is displayed.</p> <p>The modify callback function is passed the following arguments:</p> <pre>s_fieldName t_latestTextValue g_sourceOfChange</pre> <p>where <i>s_fieldName</i> is the symbol of the field in which the change was made, <i>t_latestTextValue</i> is the latest text value of the field, and <i>g_sourceOfChange</i> is <i>t</i> or <i>nil</i> where <i>t</i> indicates that the change was made programmatically (by changing the value of the field) and <i>nil</i> indicates that the change was made by a user action.</p> <p>The modify callback function should return the following:</p> <pre>t nil value</pre>

Cadence User Interface SKILL Functions Reference

Forms

If the modify callback function returns `t`, the changes made to the field are allowed and are displayed as they are entered. If the function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be an integer. This value replaces the current value of the field.

t_fieldFormat

String that can control the display of the value on the form. It is similar to `printf`.

Note: Acceptable format characters are those allowed by the C programming language.

l_fieldRange

Optional argument. It is a list containing the minimum and maximum integer values. For example, the list `(0 10)`, also specified as `0:10`, represents the range of integers from 0-10 inclusive. To specify an unbounded upper range, use either `nil` or the string `"infinity"` as the second value. To specify an unbounded lower range, use either `nil` or the string `"-infinity"` as the first value.

g_editable

If set to `nil`, the field is read-only; setting *g_editable* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` is returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#).

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_nextField

Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this

Cadence User Interface SKILL Functions Reference

Forms

argument, the Tab traversal order is determined by the *tabOrderIsAddOrder* argument of `hiCreateAppForm` or `hiCreateScrollRegion`. If the *tabOrderIsAddOrder* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Value Returned

r_fieldHandle Handle to the integer field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateLabel

```
hiCreateLabel(  
    ?name s_fieldName  
    [?labelText t_labelText]  
    [?labelIcon l_labelIcon]  
    [?justification s_justification]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?help g_fieldHelp]  
)  
=> r_fieldHandle
```

Description

Creates a standalone label entry for a form or two-dimensional menu. A `labelOnly` field contains a descriptive label. This field can provide additional features to a form or two-dimensional menu.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_labelText</i>	String which appears as the label. If you specify both <i>t_labelText</i> and <i>l_labelIcon</i> , <i>l_labelIcon</i> is used.
<i>l_labelIcon</i>	Image which appears as the label. The value of this argument must be in a valid icon format, which is a list returned by <code>getCellViewToDlist</code> , <code>dldlistToIcon</code> or <code>hiStringToIcon</code> . You can use <code>hiIsIcon</code> to check whether the icon is valid. If you specify both <i>t_labelText</i> and <i>l_labelIcon</i> , <i>l_labelIcon</i> is used.
<i>s_justification</i>	Must be one of the following symbols: 'left, 'center, or 'right. 'left is the default.
<i>t_font</i>	A font argument is accepted but not supported at this time.

Cadence User Interface SKILL Functions Reference

Forms

g_enabled *t* or *nil*. Specify *t* if you want the field enabled; specify *nil* if you want the field disabled, that is, greyed-out. The default value is *t*.

g_invisible *t* or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_fieldHelp A string or symbol used to reference help. If not specified, *s_fieldName* is used. This argument is currently not used.

Value Returned

r_fieldHandle Handle to the label.

hiCreateLayerCyclicField

```
hiCreateLayerCyclicField(  
    d_techFileId  
    t_fieldPrompt  
    t_callback  
    l_layers  
    [l_LPpair]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a cyclic field entry containing iconic representations of specified layers for a form.

Unlike other fields, you do not specify the symbol representing this field. The field will always have a field symbol name of `hiLayerField`.

Arguments

<i>d_techFileId</i>	Handle representing a techfile object.
<i>t_fieldPrompt</i>	Prompt string that appears in this field. To reference it, use <code>formHandle->hiLayerField->prompt</code>
<i>t_callback</i>	Callback string for this field. If a callback is specified, the value of the item selected is appended to this callback before being executed by SKILL. The value of this field will always be a list representing a layer icon. This value can be referenced by: <code>form->hiLayerField->value</code> The value will always be a list containing the iconic representation of the layer followed by a string representation. For example: <code>(50331991 106 26 "device (drawing1)")</code>

Cadence User Interface SKILL Functions Reference

Forms

The first three elements in this list represent the cyclic icon; the fourth element is the layer's name followed by its purpose in parentheses.

l_layers

List containing the layers that should be displayed as possible choices for this field. Each layer is represented as a list of two strings—layer name and layer purpose.

For example:

```
list( list("device" "drawing1")
      list("y0" "drawing")
      list("y1" "drawing")
      list("y2" "drawing") )
```

The layers specified must be defined in *d_techFileId*. If *nil* is specified, all layers in *d_techFileId* will be displayed.

l_LPpair

An optional argument specifying the initial value of this layer field. This argument is a list containing the layer name (or number) and purpose. This layer must exist in *d_techFileId*. If not specified, the first layer in the list *l_layers* will be the initial value.

g_invisible

t or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle

Handle to the field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateListField

```
hiCreateListField(  
    ?name s_fieldName  
    [?value l_currentValue]  
    [?help g_fieldHelp]  
    [?prompt t_fieldPrompt]  
    [?defValue l_defaultValue]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a list field entry for a form. A list field accepts a valid SKILL expression as input. SKILL's `linereadstring` determines if the expression is valid. The list expression ID is not evaluated; it is only checked to see if it is valid.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>l_currentValue</i>	Initial and current value of this field in the correct format. May be <code>nil</code> , in which case the list field appears blank. To reference the current value, use <code>formHandle->hiFieldSym->value</code>
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_fieldPrompt</i>	An optional prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>

Cadence User Interface SKILL Functions Reference

Forms

l_defaultValue Default value of the field in the correct format. This value is equal to the initial entry in *currentValue* if *defaultValue* is not specified. To reference the default value, use

`formHandle->hiFieldSym->defValue`

t_font A font argument is accepted but not supported at this time.

t_callback One or more SKILL functions to be executed if a field's value has changed. The callback is called when the field has “lost focus” (when the cursor leaves the form, the user tabs to another field, and so forth) or when the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the *t_modifyCallback* argument.

Note: If you specify multiple SKILL statements for execution, they must be enclosed in curly braces: {}.

t_modifyCallback **Note:** Do not use the *t_modifyCallback* argument with this field because it may not work as expected. Use this argument only with string fields.

SKILL function to be executed whenever any change is made in the text field, for example, by typing or pasting. The modify callback is called when a change is detected in the field but before the change is displayed.

The modify callback function is passed the following arguments:

s_fieldName *t_latestTextValue* *g_sourceOfChange*

where *s_fieldName* is the symbol of the field in which the change was made, *t_latestTextValue* is the latest text value of the field, and *g_sourceOfChange* is *t* or *nil* where *t* indicates that the change was made programmatically (by changing the `value` of the field) and *nil* indicates that the change was made by a user action.

The modify callback function should return the following:

t | *nil* | *value*

If the modify callback function returns *t*, the changes made to the field are allowed and are displayed as they are entered. If the

Cadence User Interface SKILL Functions Reference

Forms

function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be a valid SKILL expression. This value replaces the current value of the field.

g_editable

If set to `nil`, the field is read-only; setting *editable* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#).

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_nextField

Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this argument, the Tab traversal order is determined by the *tabOrderIsAddOrder* argument of `hiCreateAppForm` or `hiCreateScrollRegion`. If the *tabOrderIsAddOrder* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Value Returned

r_fieldHandle

Handle to the list field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateListBoxField

```
hiCreateListBoxField(  
    ?name s_fieldName  
    [?prompt t_prompt]  
    ?choices l_listItems  
    [?value l_selectedItem]  
    [?defValue l_defSelectedItem]  
    [?valueByPosition g_valueByPosition]  
    [?callback t_callback]  
    [?changeCB t_changeCB]  
    [?numRows x_numRows]  
    [?multipleSelect g_multipleSelect]  
    [?doubleClickCB t_doubleClickCB]  
    [?CBOOnReselect g_CBOOnReselect]  
    [?keepHistory g_keepHistory]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?help g_fieldHelp]  
)  
=> r_fieldHandle
```

Description

Creates a list box field for a form.

Any string item selection changes in this field are tracked and are accessible by SKILL.

Arguments

<i>s_fieldName</i>	Handle to the list box field.
<i>t_prompt</i>	Prompt to appear on the left side of the list box field. If no prompt is specified, none will appear.
<i>l_listItems</i>	Items displayed in the list box field. Each item is normally a null terminated string and must not contain the C newline character \n. The maximum limit on any individual string's length is 2047 bytes (2 kilobytes - 1) but strings should be kept short and concise. Default Value: nil (nothing available to select)
<i>l_selectedItem</i>	Items selected in the list box field. The items must be valid values, that is, they must have been previously defined in

Cadence User Interface SKILL Functions Reference

Forms

l_listItems. You can specify the items as a list of strings or as a list of integers or ranges representing the position of the items. For example:

```
'( "b" "e" "f" "g" "j" )  
'(2 5 6 7 10)  
'(2 '(5 7) 10)
```

The maximum limit on any individual string's length is 2047 bytes (2 kilobytes - 1).

The *g_valueByPosition* argument determines whether the *l_selectedItem* list is managed and reported as a list of strings or a list of integers. See the description of *g_valueByPosition* for more information.

Default Value: nil (nothing selected)

Note: You can also set the value with the *value* property. For example:

```
form -> listbox -> value = l_newValue
```

l_defSelectedItem

Defaults selected in the list box field. The items must be defined in *l_listItems*. You can specify the items as a list of strings or as a list of integers or ranges representing the position of the items. For example:

```
'( "b" "e" "f" "g" "j" )  
'(2 5 6 7 10)  
'(2 '(5 7) 10)
```

This argument is similar to *l_selectedItem* and is assigned to *l_defSelectedItem* when the user presses the *Defaults* button on the form.

The *g_valueByPosition* argument determines whether the *l_defSelectedItem* list is managed and reported as a list of strings or a list of integers. See the description of *g_valueByPosition* for more information.

Note: You can also set the default value with the *defValue* property. For example:

```
form -> listbox -> defValue = l_newValue
```

Cadence User Interface SKILL Functions Reference

Forms

g_valueByPosition nil or non-nil.
Default value: nil

If you set *g_valueByPosition* to nil, the items in the value and default value lists—*l_selectedItem* and *defSelectedItem*—are managed and reported as strings. When you retrieve the value or default value of the field (for example, with `form -> listbox -> value`), you get a list of strings.

If you set *g_valueByPosition* to non-nil, the items in the value and default value lists—*l_selectedItem* and *defSelectedItem*—are managed and reported as integers representing their position. The position is determined by the order of the items in the list box field—the first item is 1, the second 2, and so forth. If *g_valueByPosition* is non-nil, when you retrieve the value or default value of the field (for example, with `form -> listbox -> defValue`), you get a list of integers.

To get the field's value and default value as a list of strings even when *g_valueByPosition* is non-nil, use the [hiGetListBoxValue](#) function.

Note: You can also set the `?valueByPosition` argument with the `valueByPosition` property. For example:
`form -> listbox -> valueByPosition = nil`

t_callback SKILL command executed when the user has exited the list box field and the value has changed.

t_changeCB SKILL command executed when the value (selected items) of the list box has changed. Programmatic SKILL changes can also trigger this callback, for example:

```
form->listbox->value->list( "newValue" )
```

x_numRows Number of rows to be displayed in the list box field in a 1D form or minimum number of rows to be displayed in the list box field in a 2D form.

Note: Whenever a horizontal scrollbar appears in the field, the number of visible rows is reduced by one.

Cadence User Interface SKILL Functions Reference

Forms

g_multipleSelect Boolean (*t* or *nil*) indicator that multiple choice selections are available in this list box field. If *t* is specified, multiple items can be selected in the list box. If *nil* is specified, only single-choice selections are allowed.

Default Value: *nil*

t_doubleClickCB SKILL command executed when the user double clicks an item in the list box. The double-clicked item is selected and all previously selected values are deselected. If the selected item has changed as a result of this double click, *t_changeCB* is also called.

Default Value: *nil* (no double click occurred)

g_CBOnReselect *t* or *nil*. If you specify *t*, the change callback (*t_changeCB*) is called when a value is selected, even if it is the same value as before; otherwise, the change callback is called only when the value changes.

g_keepHistory *t* or *nil*. If you specify *t*, attempts to retain the ?value when choices are changed. The default value is *nil*.

Note: If *g_valueByPosition* is non-*nil*, the position integers are retained instead of strings. For example, if the current value is 3 and the item list is ("a" "b" "c" "d"), when you change the item list to ("c" "d" "e" "f"), the current value will remain 3 even though its corresponding string has changed from "c" to "e".

g_enabled *t* or *nil*. If you specify *t*, the field is enabled; if you specify *nil*, the field is disabled, that is, greyed-out. The default value is *t*.

g_invisible *t* or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_fieldHelp A string or symbol used to reference help. If not specified, *s_fieldName* is used. This argument is currently not used.

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

r_fieldHandle Handle to the list box field.

hiCreateMLTextField

```
hiCreateMLTextField(  
    ?name s_fieldName  
    [?prompt t_prompt]  
    [?value t_value]  
    [?defValue t_defaultValue]  
    [?font t_font]  
    [?hasVerticalScrollbar g_hasVerticalScrollbar]  
    [?hasHorizontalScrollbar g_hasHorizontalScrollbar]  
    [?enableWordWrap g_enableWordWrap]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?callback t_callback]  
    [?changeCB t_changeCB]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a multiline text field entry for a form. Any input typed into this field will be converted to a single string accessible through SKILL.

To select the proper size and location of the multiline text field in a form, first select a specific size range for the field, and then use the [hiGetTextFieldFit](#) function to find out the pixel size it needs for the specified range.

Important

To create a new line in a multiline text field, the user has to press either Control-Return or Shift-Return. Pressing just the Return key applies the changes indicated in the form. To move to the next tab stop, the user has to press Control-Tab. Pressing just the Tab key will move the cursor to the next field in the form.

Important

For a programming sample of `hiCreateMLTextField`, see ["File Browser: Creating Multiline Text Fields."](#)

Arguments

<i>s_fieldName</i>	Handle to this field within a form.
--------------------	-------------------------------------

Cadence User Interface SKILL Functions Reference

Forms

<i>t_prompt</i>	Single SKILL string indicating the prompt for the multiline text field. The prompt will be displayed on the left side of the text field.
<i>t_value</i>	Single SKILL string indicating the content of the multiline text field. The string can contain the C newline character ('\n') to indicate an advance to the next logical line. The last logical line may or may not have an ending newline. The maximum length of the string is 8191 bytes. The default value is <i>nil</i> .
<i>t_defaultValue</i>	Single SKILL string indicating the default content of the multiline text field. The value of the text field is set to <i>defaultValue</i> if the user clicks the Defaults button on a form.
<i>t_font</i>	The font to use in the multiline text field. It must be <i>nil</i> , causing the font to be derived from the defaults. The default is determined by the user's <i>.cdsinit</i> file. If there is no font specified in the user's <i>.cdsinit</i> , the "fixed" font will be used. Using a specified font name is not supported.
<i>g_hasVerticalScrollbar</i> (<i>t/nil</i>)	Indicates if there should be a vertical scroll bar displayed for the multiline text field. The default is to have a vertical scroll bar (<i>t</i>).
<i>g_hasHorizontalScrollbar</i> (<i>t/nil</i>)	Indicates if there should be a horizontal scroll bar displayed for this field. The default is to have a horizontal scroll bar (<i>t</i>). If set to <i>t</i> , there will be no word wrapping for the text field no matter what value <i>g_enableWordWrap</i> is set to.
<i>g_enableWordWrap</i> (<i>t/nil</i>)	<p>If this argument is set to <i>t</i>, and there is no horizontal scroll bar for the field, word wrapping will occur. The default is <i>nil</i>.</p> <p>Note: If <i>g_hasHorizontalScrollbar</i> is set to <i>t</i>, there will be no word wrapping even if you set <i>g_enableWordWrap</i> to <i>t</i>.</p>
<i>g_editable</i>	If set to <i>nil</i> , the field is read-only; setting <i>editable</i> to <i>t</i> makes the field editable. <i>t</i> is returned on successful completion. If an error occurs, a message is issued and <i>nil</i> returned. The default is <i>t</i> .

Cadence User Interface SKILL Functions Reference

Forms

Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of *hiCreateAppForm* is set to 'allTypein. For more information, see [hiCreateAppForm](#).

g_enabled *t* or *nil*. Specify *t* if you want the field enabled; specify *nil* if you want the field disabled, that is, greyed-out. The default value is *t*.

g_invisible *t* or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

t_callback SKILL callback to be executed when the user exits the text field.

t_changeCB SKILL callback to be executed when a significant change occurs, for example if the user clicks *Enter*, pastes a newline, or exits the field after making changes to the text. Programmatic SKILL changes can also trigger this callback. The default is *nil*.

g_nextField Symbol or string indicating the next field to bring into focus when the *Tab* key is pressed. If you specify *nil* or do not specify this argument, the *Tab* traversal order is determined by the *tabOrderIsAddOrder* argument of *hiCreateAppForm* or *hiCreateScrollRegion*. If the *tabOrderIsAddOrder* argument is not set or if its value is *nil*, the window manager's default traversal order is used.

Value Returned

r_fieldHandle Handle to the multiple-line text field.

hiCreateOptionsForm

```
hiCreateOptionsForm(  
    s_formHandle  
    t_formName  
    l_fieldEntries  
    t_help  
)  
=> r_form
```

Description

Generates a form with the specified field entries, setting it to *formHandle*.

If the form is created successfully, an internal data structure for the form is returned. Otherwise, if the arguments to *hiCreateOptionsForm* are invalid, an error message is issued and *nil* is returned. Options forms may be created using exactly the same fields as those used by standard forms.

Note: This function is only used for enterfunctions and cannot be used independently.

Arguments

<i>s_formHandle</i>	Global SKILL symbol used to reference this form. An options form is to be used with enterFunctions only.
<i>t_formName</i>	Name that appears in the banner of the form.
<i>l_fieldEntries</i>	List of field descriptors returned from the field creation routines. You can specify any number of field entries (but there must be at least one). The fields on the form appear in a column, in the order they are specified.
<i>t_help</i>	String containing a keyword that indexes into a FrameMaker help document.

Value Returned

<i>r_form</i>	Handle to the form.
---------------	---------------------

Cadence User Interface SKILL Functions Reference

Forms

hiCreatePointField

```
hiCreatePointField(  
    ?name s_fieldName  
    [?value l_currentValue]  
    [?help g_fieldHelp]  
    [?prompt t_fieldPrompt]  
    [?defValue l_defaultValue]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a point field entry for a form.

A point field accepts only a point as input. A valid point is represented by a SKILL list of two numbers (the x and y coordinates). These numbers can be floating-point numbers or integers.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>l_currentValue</i>	Initial and current value of this field in the correct format. May be <code>nil</code> , in which case the point field appears blank. To reference the current value, use <code>formHandle->hiFieldSym->value</code>
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_fieldPrompt</i>	Optional prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>

Cadence User Interface SKILL Functions Reference

Forms

l_defaultValue Default value of the field (in the correct format). This value is equal to the initial entry in *currentValue* if *defaultValue* is not specified. To reference the default value, use

`formHandle->hiFieldSym->defValue`

t_font Arguments may be accepted for some form fields but are not supported at this time.

t_callback One or more SKILL functions to be executed if a field's value has changed. The callback is called when the field has lost focus (when the cursor leaves the form, the user tabs to another field, and so forth) or when the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the *t_modifyCallback* argument.

Note: If you specify multiple SKILL statements for execution, they must be enclosed in curly braces: {}.

t_modifyCallback **Note:** Do not use the *t_modifyCallback* argument with this field because it may not work as expected. Use this argument only with string fields.

SKILL function to be executed whenever any change is made in the text field, for example, by typing or pasting. The modify callback is called when a change is detected in the field but before the change is displayed.

The modify callback function is passed the following arguments:

s_fieldName *t_latestTextValue* *g_sourceOfChange*

where *s_fieldName* is the symbol of the field in which the change was made, *t_latestTextValue* is the latest text value of the field, and *g_sourceOfChange* is *t* or *nil* where *t* indicates that the change was made programmatically (by changing the value of the field) and *nil* indicates that the change was made by a user action.

The modify callback function should return the following:

t | *nil* | *value*

Cadence User Interface SKILL Functions Reference

Forms

If the modify callback function returns `t`, the changes made to the field are allowed and are displayed as they are entered. If the function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be a point. This value replaces the current value of the field.

g_editable

If set to `nil`, the field is read-only; setting *editable* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#).

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_nextField

Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this argument, the Tab traversal order is determined by the *tabOrderIsAddOrder* argument of `hiCreateAppForm` or `hiCreateScrollRegion`. If the *tabOrderIsAddOrder* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Value Returned

r_fieldHandle

Handle to the point field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreatePointListField

```
hiCreatePointListField(  
    ?name s_fieldName  
    [?value l_currentValue]  
    [?help g_fieldHelp]  
    [?prompt t_fieldPrompt]  
    [?defValue l_defaultValue]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a point list field entry for a form.

A point list field accepts only a point list as input. A valid point list is represented by one or more points (a list of two integers or floats), separated by a blank space.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>l_currentValue</i>	Initial and current value of this field in the correct format. May be <code>nil</code> , in which case the field appears blank. To reference the default value, use <code>formHandle->hiFieldSym->value</code>
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_fieldPrompt</i>	Optional prompt string that appears in this field To reference it, use <code>formHandle->hiFieldSym->prompt</code>

Cadence User Interface SKILL Functions Reference

Forms

l_defaultValue Default value of the field in the correct format. This value is equal to the initial entry in *currentValue* if *defaultValue* is not specified. To reference the default value, use

`formHandle->hiFieldSym->defValue`

t_font An optional font argument is accepted but not supported at this time.

t_callback One or more SKILL functions to be executed if a field's value has changed. The callback is called when the field has lost focus (when the cursor leaves the form, the user tabs to another field, and so forth) or when the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the *t_modifyCallback* argument.

Note: If you specify multiple SKILL statements for execution, they must be enclosed in curly braces: {}.

t_modifyCallback **Note:** Do not use the *t_modifyCallback* argument with this field because it may not work as expected. Use this argument only with string fields.

SKILL function to be executed whenever any change is made in the text field, for example, by typing or pasting. The modify callback is called when a change is detected in the field but before the change is displayed.

The modify callback function is passed the following arguments:

s_fieldName *t_latestTextValue* *g_sourceOfChange*

where *s_fieldName* is the symbol of the field in which the change was made, *t_latestTextValue* is the latest text value of the field, and *g_sourceOfChange* is *t* or *nil* where *t* indicates that the change was made programmatically (by changing the value of the field) and *nil* indicates that the change was made by a user action.

The modify callback function should return the following:

t | *nil* | *value*

Cadence User Interface SKILL Functions Reference

Forms

If the modify callback function returns `t`, the changes made to the field are allowed and are displayed as they are entered. If the function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be a point list. This value replaces the current value of the field.

g_editable

If set to `nil`, the field is read-only; setting *editable* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#).

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_nextField

Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this argument, the Tab traversal order is determined by the *tabOrderIsAddOrder* argument of `hiCreateAppForm` or `hiCreateScrollRegion`. If the *tabOrderIsAddOrder* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Value Returned

r_fieldHandle

Handle to the point list field.

hiCreateRadioField

```
hiCreateRadioField(  
    ?name s_fieldName  
    ?choices l_radioList  
    [?help g_fieldHelp]  
    [?prompt t_fieldPrompt]  
    [?value t_currentSelection]  
    [?defValue t_defaultSelection]  
    [?itemsPerRow x_itemsPerRow]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?callback l_radioListCallbacks]  
)  
=> r_fieldHandle
```

Description

Creates a radio field for a form.

A radio field contains several radio items which default to being displayed in one row. Each radio item conveys on or off status or yes or no meaning. The items in a radio field are mutually exclusive—no more than one may be set at any time. Each radio item within the radio field may specify its own callback.

Note: If the radio field is instantiated in a one-dimensional form whose width is less than the width of the radio field, the `?itemsPerRow` argument is set automatically and is based on the width of the longest `?choices` string.

Note: The vertical spacing of multi-row toggle and radio fields in 2D forms has been changed in the IC 5.0 release. Due to a Sun bug, the spacing in 4.4.6 software running on Sun Solaris did not match the spacing in 4.4.6 software running on HP or IBM platforms. Nor did the spacing in 4.4.5 software running on Solaris 2.6, Solaris 7, or Solaris 8 match the spacing on Solaris 2.5.1. If you created 2D forms that have multi-row toggle or radio fields and you run IC 4.4.6 or earlier software on Solaris 2.6, Solaris 7, or Solaris 8, you may need to lay out the forms again to accommodate the changed spacing. This change has also been hotfixed into IC 4.4.6 in MSR3.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
--------------------	--

Cadence User Interface SKILL Functions Reference

Forms

<i>l_radioList</i>	List of strings representing the selections that can be made. These strings appear as the prompts for the radio buttons.
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_fieldPrompt</i>	Optional prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>t_currentSelection</i>	String of the radio button representing the <i>on</i> value (defaulting to the first, if not specified). Must be a string in <i>l_radioList</i> .
<i>t_defaultSelection</i>	Optional default value of the field. This value is equal to the initial entry in <i>currentSelection</i> if <i>defaultSelection</i> is not specified. To reference the default value, use <code>formHandle->hiFieldSym->defValue</code>
<i>x_itemsPerRow</i>	Optional integer argument specifying the number of toggle items to lay out in one row. The items are divided into as many rows as necessary, in row-major order, to accommodate this value. If <i>itemsPerRow</i> is 1, the buttons are arranged in a single column. Note: The toolkit imposes restrictions on the geometry allocated to these rows. Although an <i>itemsPerRow</i> value is specified, the toolkit may ignore this layout if it does not make optimum use of screen geometry.
<i>t_font</i>	An optional font argument is accepted but is not supported at this time.
<i>g_enabled</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want the field enabled; specify <i>nil</i> if you want the field disabled, that is, greyed-out. The default value is <i>t</i> .
<i>g_invisible</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want the field to be invisible; specify <i>nil</i> if you want the field to be visible. The default value is <i>nil</i> .

Cadence User Interface SKILL Functions Reference

Forms

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

l_radioListCallbacks

Optional list of user-defined SKILL procedures that are executed when the corresponding item in the *radioList* is triggered. If only one string is specified, it is called for each radio item when it is selected.

Value Returned

r_fieldHandle Handle to the radio field.

Cadence User Interface SKILL Functions Reference

Forms

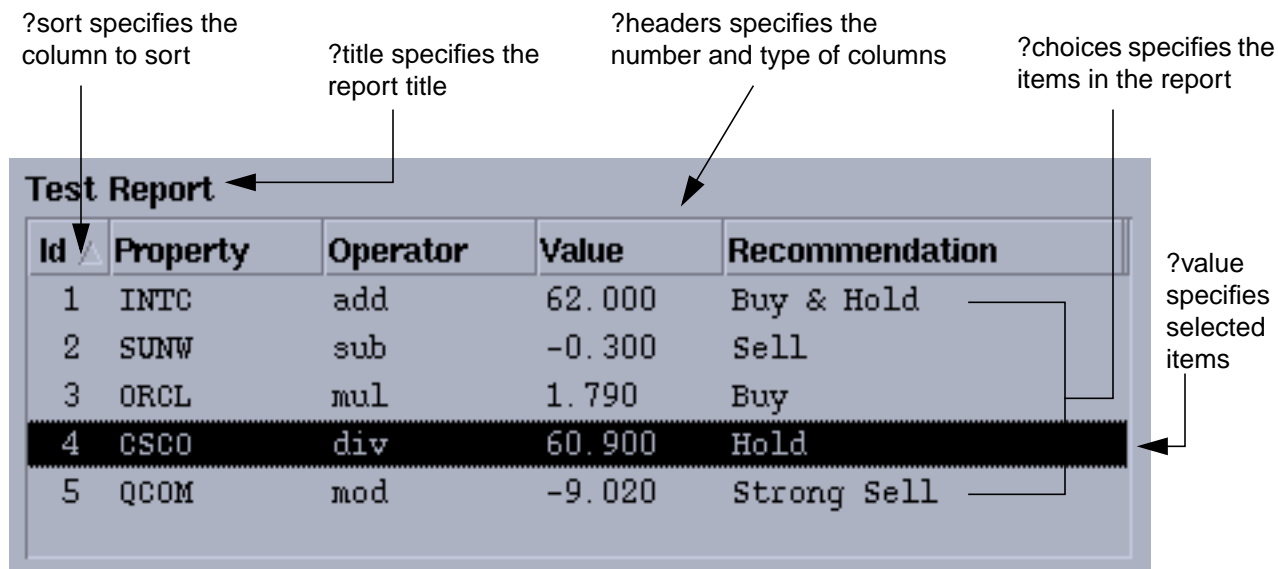
hiCreateReportField

```
hiCreateReportField(  
    ?name s_fieldName  
    [?title t_title]  
    [?titleAlignment g_titleAlignment]  
    [?headers g_headers]  
    [?choices l_choices]  
    [?value g_value]  
    [?defValue g_defaultValue]  
    [?returnPackedValue g_returnPackedValue]  
    [?help g_fieldHelp]  
    [?callback S_callback]  
    [?enableDeselectCB S_enableDeselectCB]  
    [?sortCallback S_sortCallback]  
    [?sort l_sort]  
    [?selectMode s_selectMode]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a report field, which you can use to display information in a complex format. For example, you can display a table with sortable columns. The items in a report field can be selected but they cannot be edited.

The following figure shows a report field:



Cadence User Interface SKILL Functions Reference

Forms

Arguments

s_name Handle to the field. To reference it, use

`formHandle->hiFieldSym`

t_title Report title, which is displayed above the report.

g_titleAlignment Alignment of the report title. Use one of the following values:

'left
'center
'right

The default value is 'left.

g_headers The list of columns in the report. Specify each column as a list that contains the following elements:

`'(columnName columnWidth colNameAlignment type sortable compareFunction)`

- *columnName* is the heading of the column
- *columnWidth* is the initial width of the column. The width can be changed interactively after the form is created.
- *colNameAlignment* is the alignment of the column heading and the data in the column: 'left, 'right, or 'center
- *type* is the type of data in the column: 'int, 'float[n], or 'string. You can specify float in the following ways:
 - 'float*n* (where *n* is the number of decimal points to be displayed in the float field). For example, 'float4.
 - 'float (which defaults to 'float3)
- *sortable* indicates if the column is sortable. Specify `t` if you want the column to be sortable; `nil`, otherwise. The report field can sort integer, float, and string data. Clicking on the column header sorts the data in the column.
- *compareFunction* is the name (string) of the optional sorting function you write if you want to sort the data yourself. The return value of this sorting function must be -1, 0, or 1.

Note: You need to specify *type*, *sortable*, and *compareFunction* only if you want the column to be sortable.

Cadence User Interface SKILL Functions Reference

Forms

l_choices

List containing the items in the report. Each item forms a row of the report. For example:

```
?choices list(
    list(1 "ABC" "buy" 62.0 "Buy & Hold")
    list(2 "DEF" "sell" 17.0 "Sell")
    list(3 "XYZ" "hold" 9.5 "Hold")
)
```

To retrieve or change the value of *l_choices*, use:

```
formHandle->hiFieldSym->choices
```

g_value

Initial value of the field, indicating which items in the report are selected when the form first appears.

Specify *g_value* as a list of integers, in which each integer represents an item in *?choices*. The first item in *?choices* is 0, the second item is 1, the third item is 2, and so on. You can specify a range of items as a list within the list.

For example:

```
?value '(1 3 6)           ; selects items 1, 3, 6
?value '('(2 4))          ; selects items 2, 3, 4
?value '('(1 3) 9 11)      ; selects items 1, 2, 3, 9, 11
```

After the form is instantiated, you can use the *value* property to retrieve or change the list of currently selected items. To retrieve the list of selected items, use the following syntax:

```
formHandle->hiFieldSym->value
```

The *value* property returns a list of integers. This list will contain packed values if the *g_returnPackedValue* argument is *t* (see the description of *g_returnPackedValue* for more information).

To change the list of selected items with the *value* property, use the following syntax:

```
formHandle->hiFieldSym->value = listOfItems
```

Setting the value through the *value* property always triggers the callback of the report field. If you want to set the value without triggering the callback, use the functions [*hiReportSelectItem*](#), [*hiReportSelectItems*](#), or

Cadence User Interface SKILL Functions Reference

Forms

hiReportSelectAllItems, with the *g_notify* argument set to nil.

g_defaultValue

Default value of the field, indicating which items in the report are selected when the *Defaults* button is clicked. After the form is instantiated, you can use the *defValue* property to retrieve or change the default value.

Specify *g_defaultValue* as a list of integers, in which each integer represents an item in *?choices*. The first item in *?choices* is 0, the second item is 1, the third item is 2, and so on. You can specify a range of items as a list within the list.

For example:

```
?defValue '(1 3 6)           ; selects items 1, 3, 6
?defValue '('(2 4))          ; selects items 2, 3, 4
?defValue '('(1 3) 9 11)     ; selects items 1, 2, 3, 9, 11
```

Use the following syntax for the *defValue* property

```
formHandle->hiFieldSym->defValue
```

The *defValue* property returns a list of integers. This list will contain packed values if the *g_returnPackedValue* argument is *t* (see the description of *g_returnPackedValue* for more information).

g_returnPackedValue

t or nil. Specify *t* if you want the *value*, *defValue*, and *lastValue* properties to return packed values. Otherwise, specify nil. *g_returnPackedValue* also determines whether the second argument passed to *S_callback* contains packed values.

A packed value is a range of items specified as a list within the list. For example, if items 1, 2, 3, 4, 5, 9, and 11 are selected in the report, the *value* property will return the following:

```
'('(1 5) 9 11)           ; if g_returnPackedValue is t
'(1 2 3 4 5 9 11)        ; if g_returnPackedValue is nil
```

g_fieldHelp

A string or symbol used to reference help. If not specified, *s_fieldName* is used. This argument is currently not used.

Cadence User Interface SKILL Functions Reference

Forms

S_callback

A string or a symbol that specifies the name of the callback procedure to be called when items are selected. If you want the callback procedure to also be called when items are deselected, specify `t` as the value of the `?enableDeselectCB` argument.

The callback procedure is passed two arguments. The first argument is the symbol representing the field name. The second argument is a list of integers representing the selected items of the report (the first item is 0, the second item is 1, and so forth). The second argument will contain packed values if the value of the `g_returnPackedValue` argument is `t` (see the description of `g_returnPackedValue` for more information).

For example:

```
procedure( reportCB(fieldName selectedItems "sl")
  printf( "Report field callback, fieldName = %L, selectedItems = %L"
    fieldName selectedItems )
)

reportField = hiCreateReportField(
  ?name      'reportField
  ?title     "Report Field"
  ?selectMode 'single
  ?headers   '(("Property"    120 'left)
              ("Operator"    70 'left)
              ("Value"       120 'left))
  ?value     '(("test"  "==" "argh")
              ("test2" ">" "argh")
              ("test3" "<" "test4"))
  ?callback  'reportCB
)
```

S_enableDeselectCB

`t` or `nil`. Specify `t` if you want the callback procedure (specified with the `?callback` argument) to be called when items are deselected, in addition to being called when items are selected. If you specify `nil`, the callback procedure is called only when items are selected.

The default value of *S_enableDeselectCB* is `nil`.

S_sortCallback

A string or symbol that specifies the name of the sort callback procedure to be called when a column is sorted. The sort

Cadence User Interface SKILL Functions Reference

Forms

callback procedure is passed two arguments. The first argument is the field name symbol; the second argument is `list(columnToSort sortDirection)`, where `columnToSort` is an integer indicating the column to sort (the first column is 0, the second is 1, and so on) and `sortDirection` is `t` or `nil` (`t` indicates descending order, `nil` indicates ascending order).

l_sort

Indicates which column is sorted when the form first appears. Specify *l_sort* as a list of two elements:

```
'(columnToSort sortDirection)
```

where `columnToSort` is an integer indicating the column to sort (the first column is 0, the second is 1, and so on) and `sortDirection` is `t` or `nil` (`t` indicates descending order, `nil` indicates ascending order).

You can also use the `sort` property to sort a column after the form is instantiated.

Use the following format for the `sort` property:

```
formHandle->hiFieldSym->sort
```

Clicking on the column header sorts the column. Clicking again on the column header changes the sorting direction (ascending or descending).

s_selectMode

Specifies the selection mode for the report field. Use one of the following symbols:

- | | |
|-----------|---|
| 'single | Only one item can be selected at a time. Click to select the item.

To deselect an item, click it again or select another item. |
| 'multiple | Multiple items can be selected. Click to select the items.

To deselect an item, click it again. |

Cadence User Interface SKILL Functions Reference

Forms

<code>'browse</code>	<p>One item always remains selected. Click or drag to select the item.</p> <p>To deselect an item, select another item.</p>
<code>'extended</code>	<p>One or more items always remain selected. Multiple contiguous ranges of items can be selected. Click to select a single item. Click and drag to select a range of items. Press the <code>CTRL</code> key and click to select an additional item or a range of additional items.</p> <p>To deselect an item, press the <code>CTRL</code> key and click on the item. <code>CTRL+click</code> also deselects the last selected item.</p> <p>You can also use <code>CTRL+SHIFT</code> to select or deselect a range of items.</p>

The default mode is `'extended`.

You can modify the value of this argument with the `selectMode` property. For example:

```
formHandle->fieldSym->selectMode = 'single
```

`g_invisible`

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

`g_nextField`

Symbol or string indicating the next field to bring into focus when the `Tab` key is pressed. If you specify `nil` or do not specify this argument, the `Tab` traversal order is determined by the `tabOrderIsAddOrder` argument of [hiCreateAppForm](#) or [hiCreateScrollRegion](#). If the `tabOrderIsAddOrder` argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

r_fieldHandle Handle to the report field.

Reference

[hiReportSelectItem](#), [hiReportSelectItems](#), [hiReportSelectAllItems](#),
[hiReportDeselectItem](#), [hiReportDeselectItems](#), [hiReportDeselectAllItems](#),
[hiReportGetSelectedItems](#)

Cadence User Interface SKILL Functions Reference

Forms

hiCreateScaleField

```
hiCreateScaleField(  
    ?name s_fieldName  
    [?prompt t_prompt]  
    [?value x_currentValue]  
    [?callback t_callback]  
    [?isContinuous g_isContinuous]  
    [?range l_range]  
    [?defValue x_defaultValue]  
    [?help g_fieldHelp]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a scale field for a form.

A scale field indicates a value from within a range of numerical values and allows the user to input or modify a value from the same range. The field consists of a rectangular region containing a slider indicating the current value. It also includes a label indicating the scale value at any position of the slider.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_prompt</i>	Prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>x_currentValue</i>	Initial and current value of the scale field which must be within the given range <code>formHandle->hiFieldSym->value</code>
<i>t_callback</i>	One or more SKILL functions to be executed if a field's value has changed. Called when the slider is released in a new position or when it moves incrementally along the bar, depending on the

Cadence User Interface SKILL Functions Reference

Forms

value of the *g_isContinuous* argument. If *g_isContinuous* is not specified, the callback is executed when the slider is released in the new position.

Note: To specify multiple SKILL statements for execution, they must be enclosed by curly braces: {}.

g_isContinuous *t* or *nil*. When *g_isContinuous* is *t*, the callback is continuous while the slider is being moved; when *g_isContinuous* is *nil*, the callback is executed when the slider is released in the new position. The default value is *nil*. You can modify *g_isContinuous* at any time with the following property:

```
formHandle->hiFieldSym->isContinuous = t|nil
```

l_range Indicates the upper and lower values the scale field may obtain. The default range is from 0 to 1000, inclusive. To change the range the slider displays, the user may type
`form->scalefield->range = newrange`
where *newrange* is a list of two numbers. These numbers can include negative integers.

If the maximum value for the range is not an integer, it will be set to 1000. If the minimum value for the range is not an integer or is equal to or greater than the maximum value, it will be set to the maximum value minus 1000 (*maxValue* - 1000).

If the range is changed, *currentValue* and *defaultValue* are modified, if necessary, to fall within the new range.

x_defaultValue Default value of the field (in the correct format). This value is equal to the initial entry in *currentValue* if *defaultValue* is not specified. To reference it, use

```
formHandle->hiFieldSym->defValue
```

g_fieldHelp A string or symbol used to reference help. If not specified, *s_fieldName* is used. This argument is currently not used.

t_font An optional font argument is accepted but is not supported at this time.

Cadence User Interface SKILL Functions Reference

Forms

g_enabled `t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible `t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle Handle to the scale field.

hiCreateSeparatorField

```
hiCreateSeparatorField(  
    ?name s_fieldName  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a separator field for a form.

A separator is a line used to separate fields for easier visual recognition. It is always two pixels thick. Only horizontal separators can be created for one-dimensional forms. Two-dimensional forms can have vertical or horizontal separators. A horizontal separator in a 1D form always spans the width of the form.

When specifying 2D attributes for a separator field use the following format:

```
list( r_separatorField x_xpos:x_ypos x_length:x_orient )
```

x_xpos:*x_ypos* represents the origin of the separator. The length can be controlled by *x_length* but the thickness is always two pixels. *x_orient* is 0 if the separator is horizontal, 1 if vertical.

Arguments

<i>s_fieldName</i>	Handle to this field within a form.
<i>g_invisible</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want the field to be invisible; specify <i>nil</i> if you want the field to be visible. The default value is <i>nil</i> .

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

<i>r_fieldHandle</i>	Handle for the separator field.
----------------------	---------------------------------

Cadence User Interface SKILL Functions Reference

Forms

Example

An example of specifying a horizontal separator 40 pixels long starting at 10:30 is:

```
list( hiCreateSeparatorField proglid( ?name 'sep) 10:30 40:0)
```

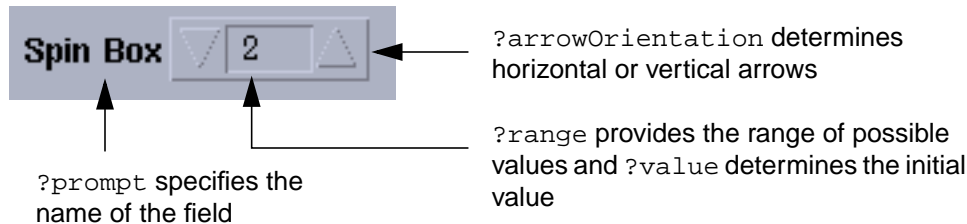
hiCreateSpinBox

```
hiCreateSpinBox(  
    ?name s_fieldName  
    [?prompt t_prompt]  
    [?help g_fieldHelp]  
    [?value g_value]  
    [?defValue g_defaultValue]  
    [?font t_fontName]  
    [?callback t_callback]  
    [?range l_rangeOfItems]  
    [?arrowOrientation g_arrowOrientation]  
    [?shadowed g_shadowed]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
)  
=> r_fieldHandle
```

Description

Creates a spin box field, which contains a range of items that users can cycle through. The spin box field can only contain integers.

The following figure displays a spin box field:



Arguments

s_fieldName Handle to the spin box field. To reference it, use

`formHandle->hiFieldSym`

t_prompt The name that appears next to the spin box field, specified as a string.

Cadence User Interface SKILL Functions Reference

Forms

<i>g_fieldHelp</i>	A string, symbol, list of strings, or list of symbols used to reference help. If this argument is not specified, <i>s_name</i> is used. This argument is not currently used.
<i>g_value</i>	The initial and current value of the field. <i>g_value</i> must be an integer within the range specified by <i>l_rangeOfItems</i> .
<i>g_defaultValue</i>	The default value of the field. <i>g_defaultValue</i> must be an integer within the range specified by <i>l_rangeOfItems</i> .
<i>t_fontName</i>	An optional font argument is accepted but is not supported at this time.
<i>t_callback</i>	One or more SKILL functions to be executed when the field's value has changed. If you specify multiple SKILL statements, they must be enclosed by curly braces: <code>{ }</code> .
<i>l_rangeOfItems</i>	A list of two integers defining the range of values in the spin box field. The first integer must be less than the second integer.

For example:

```
?range list(-12 12)
```

g_arrowOrientation

Determines the orientation of the arrows that are used to cycle through items in the spin box field: 'vertical or 'horizontal. The default value is 'horizontal.

vertical arrows



horizontal arrows



g_shadowed

t or *nil*. Specify *t* if you want a shadow box around the spin box field; *nil*, otherwise. The default value is *t*.

shadowed spin box



spin box without shadow



Cadence User Interface SKILL Functions Reference

Forms

g_enabled `t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible `t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the `invisible` property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle Handle to the spin box field.

Cadence User Interface SKILL Functions Reference

Forms

hiCreateStringField

```
hiCreateStringField(  
    ?name s_fieldName  
    [?prompt t_fieldPrompt]  
    [?value t_currentValue]  
    [?help g_fieldHelp]  
    [?defValue t_defaultValue]  
    [?font t_font]  
    [?callback t_callback]  
    [?modifyCallback t_modifyCallback]  
    [?format t_fieldFormat]  
    [?editable g_editable]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?nextField g_nextField]  
)  
=> r_fieldHandle
```

Description

Creates a string field entry for a form. Any input typed into this field is surrounded by double quotation marks before being passed to SKILL to interpret.

A string field is a field allowing for typed input. Up to three methods of encouraging the desired format of the typed input may be specified: An initial string can indicate the format of the string to be entered, the prompt can convey the type of response desired, or a format string can be specified which is automatically imposed on any user input.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>t_fieldPrompt</i>	Optional prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>t_currentValue</i>	Initial and current value of this field. To reference it, use <code>formHandle->hiFieldSym->value</code>

Cadence User Interface SKILL Functions Reference

Forms

<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>t_defaultValue</i>	<p>Optional, default value of the field. This value is equal to the initial entry in <i>currentValue</i> if <i>defaultValue</i> is not specified. To reference the value, use</p> <pre>formHandle->hiFieldSym->defValue</pre>
<i>t_font</i>	An optional font argument is accepted but is not supported at this time.
<i>t_callback</i>	<p>One or more SKILL functions to be executed if a field's value has changed. May always be specified (except for non-active label fields). For type-in fields, the callback is called when the field has "lost focus" (when the cursor leaves the form, the user tabs to another field, and so forth) or when the user selects any button in the form. If you want a callback that is called immediately, as each character is entered, use the <i>t_modifyCallback</i> argument.</p> <p>Note: If you specify multiple SKILL statements for execution, they must be enclosed by curly braces: {}.</p>
<i>t_modifyCallback</i>	<p>SKILL function to be executed whenever any change is made in the text field, for example, by typing or pasting. The modify callback is called when a change is detected in the field but before the change is displayed.</p> <p>The modify callback function is passed the following arguments:</p> <pre>s_fieldName t_latestTextValue g_sourceOfChange</pre> <p>where <i>s_fieldName</i> is the symbol of the field in which the change was made, <i>t_latestTextValue</i> is the latest text value of the field, and <i>g_sourceOfChange</i> is <i>t</i> or <i>nil</i> where <i>t</i> indicates that the change was made programmatically (by changing the value of the field) and <i>nil</i> indicates that the change was made by a user action.</p> <p>The modify callback function should return the following:</p> <pre>t nil value</pre>

Cadence User Interface SKILL Functions Reference

Forms

If the modify callback function returns `t`, the changes made to the field are allowed and are displayed as they are entered. If the function returns `nil`, the changes made to the field are not allowed and are not displayed—the original value of the field is retained. If the function returns some other value, it must be a string. This value replaces the current value of the field.

t_fieldFormat

Optional string that can control the display of the value on the form. It is similar to `printf`.

Note: Acceptable format characters are those allowed by the SKILL programming language.

g_editable

If set to `nil`, the field is read-only; setting *editable* to `t` makes the field editable. `t` is returned on successful completion. If an error occurs, a message is issued and `nil` returned. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#).

g_enabled

`t` or `nil`. Specify `t` if you want the field enabled; specify `nil` if you want the field disabled, that is, greyed-out. The default value is `t`.

g_invisible

`t` or `nil`. Specify `t` if you want the field to be invisible; specify `nil` if you want the field to be visible. The default value is `nil`.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

g_nextField

Symbol or string indicating the next field to bring into focus when the Tab key is pressed. If you specify `nil` or do not specify this argument, the Tab traversal order is determined by the *tabOrderIsAddOrder* argument of `hiCreateAppForm` or `hiCreateScrollRegion`. If the *tabOrderIsAddOrder* argument is not set or if its value is `nil`, the window manager's default traversal order is used.

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

r_fieldHandle Handle to the string field.

hiCreateTabField

```
hiCreateTabField(
    ?name s_name
    ?fields l_fieldEntries
    [?attachmentList l_fieldAttachments]
    [?tabs g_tabs]
    [?value g_currentValue]
    [?defValue g_defaultValue]
    [?tabPlacement g_tabPlacement]
    [?tabOrderIsAddOrder g_tabOrderIsAddOrder]
    [?callback g_callback]
    [?pageScroller g_pageScroller]
    [?notebookStyle g_notebookStyle]
    [?help g_fieldHelp]
    [?invisible g_invisible]
)
=> r_fieldHandle
```

Description

Creates a tab field, which can have multiple pages. You can change the pages interactively by selecting the tab attached to each page, or programmatically by using the `form->field->value` property.

Arguments

<i>s_name</i>	Handle to the tab field within a form. To reference it, use <code>formHandle->hiFieldSym</code>
<i>l_fieldEntries</i>	A list of 2-D lists. Each 2-D list forms a page of the tab field. The number and order of elements in the <i>l_fieldEntries</i> argument determine the number and order of pages in the tab field. The default value of this argument is <code>nil</code> . For more information about 2-D lists, see “Two Dimensional Form Layout” on page 212.
<i>l_fieldAttachments</i>	Determines how each page of the tab field attaches to the tab pane. For more information about field attachments, see “Field Attachments” on page 214.

Cadence User Interface SKILL Functions Reference

Forms

<i>g_tabs</i>	<p>List of strings representing the tab labels. <i>nil</i> and " " (empty string) are legal values. If you specify <i>nil</i>, the page does not have a tab; if you specify " ", the page has an empty tab. The default value of this argument is <i>nil</i>.</p>
<i>g_currentValue</i>	<p>Initial and current value of the field. Determines which page is displayed when the tab field first appears. The legal values for this argument are 1 to <i>number_of_pages</i>, where 1 indicates the first page. To change the value, use</p> <pre>form->field->value = x</pre> <p>where <i>x</i> is any number between 1 and <i>number_of_pages</i>.</p>
<i>g_defaultValue</i>	<p>Default value of the field. Determines which page of the tab field is displayed as the default. The legal values for this argument are 1 to <i>number_of_pages</i>, where 1 indicates the first page. To change the value, use</p> <pre>form->field->defValue = x</pre> <p>where <i>x</i> is any number between 1 and <i>number_of_pages</i>.</p>
<i>g_tabPlacement</i>	<p>Location of the tabs. Specify one of the following:</p> <ul style="list-style-type: none">'top'bottom'right'left <p>The default value of this argument is 'bottom.</p>
<i>g_tabOrderIsAddOrder</i>	<p><i>t</i> or <i>nil</i>. Determines the order in which pages within the tab field are traversed when the Tab key is pressed. If you specify <i>t</i>, the <i>nextField</i> property is automatically set for the fields and the traversal order is the order in which fields are added to the form (that is, the order in which they are listed in the <i>l_fieldEntries</i> list). If you specify <i>nil</i>, the window manager's default traversal order is used.</p> <p>The default value of <i>g_tabOrderIsAddOrder</i> is <i>nil</i>.</p>

Cadence User Interface SKILL Functions Reference

Forms

Note: If the *nextField* property of a field is already set to anything except *nil*, the value of *g_tabOrderIsAddOrder* does not override it.

g_callback

Callback function that will be triggered when the pages are changed. The default value of this argument is an empty string ("").

g_pageScroller

t or *nil*. Specify *t* if you want a spin box (a box that lets you scroll through the pages) to be placed in the lower right corner of the tab field. Specify *nil* if you do not want to use a spin box with the tab field. The default value of this argument is *nil*.

g_notebookStyle

t or *nil*. Specify *t* if you want to display the tab field in the Motif Notebook style. The default value of this argument is *nil*.

g_fieldHelp

A string or symbol used to reference help. If not specified, *s_name* is used. This argument is currently not used.

g_invisible

t or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

Value Returned

r_fieldHandle

Handle to the tab field.

hiCreateToggleField

```
hiCreateToggleField(  
    ?name s_fieldName  
    ?choices l_toggleList  
    [?help g_fieldHelp]  
    [?numSelect x_numSelectable]  
    [?prompt t_fieldPrompt]  
    [?value l_currentValues]  
    [?defValue l_defaultValues]  
    [?itemsPerRow x_itemsPerRow]  
    [?font t_font]  
    [?enabled g_enabled]  
    [?invisible g_invisible]  
    [?callback l_toggleListCallbacks]  
)  
=> r_fieldHandle
```

Description

Creates a toggle field entry for a form.

A toggle field may contain several toggle items which default to being displayed in one row. Each toggle item conveys on or off status, or yes or no meaning. A *numSelectable* argument is optional with toggle fields. This is the maximum number of toggle items that can be simultaneously “turned on.” If *numSelectable* is set to 1 (implying mutual exclusion), a radio type should be used instead. Each toggle item within the toggle field may specify its own callback.

Note: If the toggle field is instantiated in a one-dimensional form whose width is less than the width of the toggle field, the *?itemsPerRow* argument is set automatically and is based on the width of the longest *?choices* string.

Note: The vertical spacing of multi-row toggle and radio fields in 2D forms has been changed in the IC 5.0 release. Due to a Sun bug, the spacing in 4.4.6 software running on Sun Solaris did not match the spacing in 4.4.6 software running on HP or IBM platforms. Nor did the spacing in 4.4.5 software running on Solaris 2.6, Solaris 7, or Solaris 8 match the spacing on Solaris 2.5.1. If you created 2D forms that have multi-row toggle or radio fields and you run IC 4.4.6 or earlier software on Solaris 2.6, Solaris 7, or Solaris 8, you may need to lay out the forms again to accommodate the changed spacing. This change has also been hotfixed into IC 4.4.6 in MSR3.

Arguments

<i>s_fieldName</i>	Handle to this field within a form. To reference it, use
--------------------	--

Cadence User Interface SKILL Functions Reference

Forms

	<code>formHandle->hiFieldSym</code>
<i>l_toggleList</i>	List of symbol/string pairs (in a list), representing handles to individual toggle items within the field, and their prompts. If, instead of specifying a list of symbol/string pairs, the user decides to specify a list containing only the symbol for a toggle item, the prompt for the toggle item defaults to the print string of this symbol.
<i>g_fieldHelp</i>	A string or symbol used to reference help. If not specified, <i>s_fieldName</i> is used. This argument is currently not used.
<i>x_numSelectable</i>	Total number of choices (in <i>toggleList</i>) that may be selected at one time. This argument defaults to allowing all items to be selected.
<i>t_fieldPrompt</i>	Optional prompt string that appears in this field. To reference it, use <code>formHandle->hiFieldSym->prompt</code>
<i>l_currentValues</i>	List where each entry is either <code>t</code> or <code>nil</code> , representing the on or off value of each choice in <i>toggleList</i> . If <i>currentValues</i> is not specified, this argument defaults to a list in which every entry is <code>nil</code> . To reference the current value, use <code>formHandle->hiFieldSym->value</code> For toggle fields, an additional level of values may be obtained. The user may either inquire about all toggle values at once (a list) as shown above, or the value of an individual toggle item: <code>formHandle->hiFieldSym->toggleItemSym->value</code> (returns <code>t</code> or <code>nil</code> for the specified toggle item)
<i>l_defaultValues</i>	Optional default value of the field. This value is equal to the initial entry in <i>currentValues</i> if <i>defaultValues</i> is not specified. To reference the default value, use <code>formHandle->hiFieldSym->defValue</code>

Cadence User Interface SKILL Functions Reference

Forms

x_itemsPerRow

Row arrangement. This argument is useful for customizing the layout of toggle fields. The items are divided into as many rows as necessary, in row-major order, to accommodate this value. If *itemsPerRow* is 1, the buttons are arranged in a column.

Note: The toolkit imposes restrictions on the geometry allocated to these rows. Although an *itemsPerRow* value is specified, the toolkit may ignore this layout if it does not make optimum use of screen geometry.

t_font

An optional font argument is accepted but is not supported at this time.

g_enabled

t or *nil*. Specify *t* if you want the field enabled; specify *nil* if you want the field disabled, that is, greyed-out. The default value is *t*.

g_invisible

t or *nil*. Specify *t* if you want the field to be invisible; specify *nil* if you want the field to be visible. The default value is *nil*.

You can modify the value of this argument (after the form is created) with the *invisible* property. For example:

```
formHandle->fieldSym->invisible = nil
```

l_toggleListCallbacks

Optional list of callback strings. If only one string is specified, it is called for each toggle item, when it changes.

Value Returned

r_fieldHandle

Handle to the toggle field.

Cadence User Interface SKILL Functions Reference

Forms

hiDeleteField

```
hiDeleteField(  
    r_form  
    s_field  
)  
=> t / nil
```

Description

Deletes a field from a form.

If more than one field will be deleted from the same form, use [hiDeleteFields](#). This is especially true if a contiguous set of one-dimensional form fields will be deleted.

Deleting a field from a 2D form removes the instance of the field from the form, and assumes that there is only one instance of this field contained within the form. The field still exists in any other form which contains it.

In a two-dimensional form or menu, no geometry management is performed to “fill” the freed area of the screen. If the field is deleted from a one-dimensional form, the field is removed and the other fields adjusted appropriately.

Arguments

<i>r_form</i>	Form created by a call to <u>hiCreateAppForm</u> or <u>hiCreateForm</u> .
<i>s_field</i>	SKILL symbol representing the field in <i>r_form</i> to be deleted. This symbol is the field handle specified when the field was created.

Value Returned

<i>t</i>	Returns <i>t</i> if the field is deleted.
<i>nil</i>	Returns <i>nil</i> and an error message if the field is not added.

hiDeleteFields

```
hiDeleteFields(  
    r_form  
    l_fields  
)  
=> t / nil
```

Description

Deletes a field from a form.

If more than one field will be deleted from the same form, this function should be used instead of multiple calls to `hiDeleteField`.

For one-dimensional forms, `hiDeleteFields` optimizes the deleting of a contiguous list of fields (given in top-to-bottom order) by moving each remaining field once, after all fields have been deleted.

Deleting a field from a 2D form removes the instance of the field from the form, and assumes that there is only one instance of this field contained within the form. The field still exists in any other form which contains it.

In a two-dimensional form or menu, no geometry management is performed to “fill” the freed area of the screen. If a field is deleted from a one-dimensional form, the field is removed and the other fields adjusted appropriately.

Arguments

<i>r_form</i>	Form created by a call to <code>hiCreateAppForm</code> or <code>hiCreateForm</code> .
<i>l_fields</i>	List of SKILL symbols representing the fields in <i>r_form</i> to be deleted. These symbols are the field handles specified when the fields were created.

Value Returned

<i>t</i>	Returns <i>t</i> if the field is deleted.
<i>nil</i>	Returns <i>nil</i> and an error message if the field is not deleted.

Cadence User Interface SKILL Functions Reference

Forms

hiDeleteForm

```
hiDeleteForm(  
    r_form  
)  
=> t / nil
```

Description

Deletes the form specified and any fields the form contains.

The global SKILL handle representing this form is reset to `nil`. Users should not programmatically reset a form SKILL handle to `nil`. This function should not be called within the form's callback routine, or if the form is currently being displayed. Users should take care to delete any unnecessary forms. This routine returns `t` if the form was deleted successfully; otherwise, if this function was called from within the form's callback procedure, or the form is still displayed on the screen, `nil` is returned.

Arguments

<i>r_form</i>	Form to be deleted. It must be a valid form description returned from <code>hiCreateAppForm</code> , <code>hiCreateForm</code> , or <code>hiCreateOptionsForm</code> .
---------------	--

Value Returned

<i>t</i>	Returns <code>t</code> if the form is deleted.
<i>nil</i>	Returns <code>nil</code> and displays an error message if an error occurs.

Cadence User Interface SKILL Functions Reference

Forms

hiDisplayForm

```
hiDisplayForm(  
    r_form  
    [l_location]  
)  
=> t / nil
```

Description

Displays a form. Do not use `hiDisplayForm` for displaying options forms. You can also use this function to bring a form to the top of the screen if the form is covered by other windows.

Arguments

r_form

Form handle. It must be a valid form description returned from [hiCreateAppForm](#) (preferably) or `hiCreateForm`.

l_location

This argument will be ignored if the form placement style is anything other than the default. If specified, the form is displayed with its upper left corner at the given location. If no location is specified, the form first appears at a default position set by `hiSetFormPosition`, if previously specified (again, only if the form placement style is the default). Once a form is displayed, it will subsequently appear in the position it last appeared on the screen.

The default value can be set in your `.Xdefaults` file with the line:

```
Opus.formPlacement: default
```

or by setting the `formPlacement` property of the Command Interpreter Window to "default" with the line:

```
hiGetCIWindow()->formPlacement="default"
```

Value Returned

t

Returns `t` if the form is displayed.

nil

Returns `nil` if you press *Cancel* to bring the form down.

Cadence User Interface SKILL Functions Reference

Forms

Note: If you set the *g_dontBlock* argument of the `hiCreateAppForm` function set to `t`, a call to `hiDisplayForm()` will immediately return `t`. For more information, see [hiCreateAppForm](#).

Cadence User Interface SKILL Functions Reference

Forms

hiEditPropList

```
hiEditPropList(  
    g_object  
    [t_title]  
    [g_modifiable]  
)  
=> t / nil
```

Description

Invokes the property list editor for the given *g_object*.

The object must be either a database object, a design manager object, a probe object, or a list. This function is compatible with the Edge SKILL function `makeForm`, where *g_object* is a list.

This function can also be used to create a form by specifying a list as the *g_object*. This list must contain one or more lists of a specified format. (This format is the same format specified for the Edge SKILL function `makeForm`.) Each sublist contains a variable that corresponds to a field on the form. When the form is applied, the new values are assigned to their respective variables, so they can be used later in a SKILL procedure.

Each of the sublists that make up the main list must be in the following format:

```
'(s_sym t_prompt t_type [g_value] [l_range] [t_help] [l_constraints])
```

The elements of each sublist are:

<i>s_sym</i>	Variable (symbol) representing the field on the form.
<i>t_prompt</i>	Prompt string describing the variable's function. This prompt is displayed to the left of the corresponding field on the form.
<i>t_type</i>	String describing the type of the variable. Permitted values are: "integer" or "int" "float" "string" "boolean" or "Boolean" or "yesOrNo" "filename", "fileName" or "file"
<i>g_value</i>	Optional initial value for the variable (which must be compatible with the type specified). This value is the field's initial value when the form is first displayed. Valid values for a Boolean type are <i>t</i> ,

Cadence User Interface SKILL Functions Reference

Forms

"TRUE", "true", "yes", "FALSE", "false", "no", or nil. It is not possible to show an empty value for integer or float types, but strings can have a value of "". If an initial value is not specified, the symbol *s_sym* is evaluated for the initial value. If *s_sym* is unbound, a warning is issued and this function returns nil.

l_range

Optional list defining the range of possible values that the variable can assume. This is valid for all types except "boolean". If *t_type* is "int" or "float", *l_range* must be a list of two integers or floating point numbers that represent the minimum and maximum values of the variable. If *t_type* is "string" or "filename", *l_range* must be a list of possible strings.

t_help

Optional help string for this variable. This is currently not supported.

l_constraints

Optional string or list of strings describing possible constraints on the variable. Currently, the only constraint supported is "read-only." If this constraint is specified, the associated field associated appears greyed-out and is non-editable.

Note: This constraint is ignored if *t_type* is "boolean" or if *t_type* is "filename" or "string" and an *l_range* is specified.

Arguments

g_object

Database object, design manager object, probe object, or list.

t_title

Name to be displayed in the window manager frame of the Property List Editor form. If it is not specified, an appropriate title is used.

g_modifiable

Either *t* or nil to determine if properties can be modified. If *g_modifiable* is set to *t*, you are allowed to add, delete, or modify properties by selecting the appropriate button on the banner. If set to nil, you are not allowed to add, delete, or modify the properties in any way, except to change their values. The default for *g_modifiable* is *t*.

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

<code>t</code>	Returns <code>t</code> after the form has been brought down by pressing the Done button.
<code>nil</code>	Returns <code>nil</code> if the <code>Cancel</code> button was pressed, if <code>s_sym</code> is unbound, or if <code>g_object</code> is not a valid database object, design manager object, or list.

Example

The following is an example of `hiEditPropList` using the `list` object:

```
age = 3
mylist = list( ('myname "Name" "string" "Dumbo"
  list("Dumbo" "Mickey Mouse" "Donald Duck") )
  list('age "Age" "int")
  list('weight "Weight" "float" 3600.0
    list(0 9000.0))
  list('type "Type" "string" "ANIMAL" nil
    "" "read-only")
  ('cartoon "Disney character?" "boolean" t))
hiEditPropList( mylist "Cartoon Character" nil )
```

The symbol variables `myname`, `age`, `weight`, `type`, and `cartoon` can now be manipulated in subsequent lines. They will contain the values you specify on the form.

hiEscapeStringChars

```
hiEscapeStringChars(  
    S_stringOrSymbol  
)  
=> s_stringResult
```

Description

Escapes certain characters (backslash and double quote characters) in a string by preceding them with a backslash so that they do not cause errors when the string is evaluated in SKILL. You can use this function to prepare strings that will be evaluated in callbacks.

Arguments

S_stringOrSymbol String or symbol.

Return Value

s_stringResult String that will preserve backslash and double quote characters when it is evaluated in SKILL.

hiFormApply

```
hiFormApply(  
    r_form  
)  
=> t
```

Description

Performs the same action as would be performed by selecting the *Apply* button on the form.

This function takes the form as an argument and executes the Done callback associated with the form without unmapping it. This can either be a symbol or a string. If it is a symbol, it is the name of the function that should be called with the form as an argument. You can call

```
hiInFormApply( form ) => t/nil
```

in the Done callback to find out if the Done callback is called due to an *OK* or an *Apply* of the form.

For more information about the *Apply* button, see [hiCreateAppForm](#).

Arguments

<i>r_form</i>	Specifies which form is applied.
---------------	----------------------------------

hiFormCancel

```
hiFormCancel(  
    r_form  
)  
=> t
```

Description

Performs the same action as would be performed by selecting the *Cancel* button on the form.

If the form is mapped, it will be unmapped. The values of the form fields will be restored to the last values saved, and any field callbacks will be called if this results in a change of field values.

If `hiFormCancel` is called on a form that has a *Close* button but does not have a *Cancel* button, `hiFormCancel` will call [hiFormClose](#).

Arguments

<i>r_form</i>	Form to be cancelled.
---------------	-----------------------

Cadence User Interface SKILL Functions Reference

Forms

hiFormClose

```
hiFormClose(  
    r_form  
)  
=> t
```

Description

Equivalent to clicking the *Close* button on a form. This function is an alias for `hiFormDone`. The *Close* button is identical to the *OK* button, except that it triggers `hiFormClose` instead of `hiFormDone`.

Arguments

<i>r_form</i>	Form handle.
---------------	--------------

Value Returned

t	Returns t.
---	------------

Reference

[hiFormApply](#), [hiFormCancel](#), [hiFormDefaults](#), [hiFormDone](#), [hiFormFinish](#)

Cadence User Interface SKILL Functions Reference

Forms

hiFormDefaults

```
hiFormDefaults(  
    r_form  
)  
= t / nil
```

Description

Function logged when the *Defaults* button is pressed on a form. Can be called directly from SKILL to simulate pressing *Defaults*.

Arguments

<i>r_form</i>	Form handle.
---------------	--------------

Value Returned

t	Returns t unless the <i>Defaults</i> button is disabled.
---	--

nil	Returns nil when the Defaults button is disabled.
-----	---

Reference

[hiFormApply](#), [hiFormCancel](#), [hiFormDone](#)

Cadence User Interface SKILL Functions Reference

Forms

hiFormDone

```
hiFormDone(  
    r_form  
)  
=> t
```

Description

Performs the same action as would be performed by selecting the *OK* button on the form. This function is identical to the [hiFormFinish](#) function.

If the form is mapped, it will be unmapped according to the *earlyUnmap* setting when the form was created. If the callback status of the form is *t*, the values of the form fields are saved.

Arguments

<i>r_form</i>	Specifies the form.
---------------	---------------------

Cadence User Interface SKILL Functions Reference

Forms

hiFormFinish

```
hiFormFinish(  
    r_form  
)  
=> t
```

Description

Equivalent to pressing the *OK* button on a form. This function is identical to the [hiFormDone](#) function.

If the form is mapped, it will be unmapped according to the *earlyUnmap* value set when the form was created. If the callback status of the form is *t*, the values of the form fields are saved.

Arguments

<i>r_form</i>	Handle to the form.
---------------	---------------------

Value Returned

<i>t</i>	Returns <i>t</i> .
----------	--------------------

Cadence User Interface SKILL Functions Reference

Forms

hiFormList

```
hiFormList( )  
=> l_formSymbols
```

Description

Returns a list of all form symbols currently defined in SKILL space.

Arguments

None.

Value Returned

<i>l_formSymbols</i>	List of all form symbols currently defined in the SKILL space.
----------------------	--

Cadence User Interface SKILL Functions Reference

Forms

hiFormUnmap

```
hiFormUnmap(  
    r_form  
)  
=> t / nil
```

Description

Unmaps the form from the screen before a callback is completed.

This function can be used with [hiSetCallbackStatus](#) if the *unmapAfterCB* argument to [hiCreateAppForm](#) has been set to *t*. It unmaps the form only if the callback status of the form is *t* and if it is called within the form's Done callback. This function is especially useful for lengthy callback routines that must do error checking.

Arguments

r_form Form to be unmapped.

Value Returned

t Returns *t* if the form is unmapped.

nil Returns *nil* if *r_form* is not a form created by [hiCreateAppForm](#).

Example

This example creates a form that prompts the user for a file name. The form callback immediately unmaps the form before processing the specified file, if the file name is valid.

```
; First, create the form.  
myForm = hiCreateAppForm(?name 'myForm ?formTitle "title" ?callback 'myCB ?fields  
  
list(hiCreateStringField(?name 'fileName ?prompt "Enter filename:"))  
?help "" ?unmapAfterCB t)  
  
; myCB - form callback procedure  
procedure(myCB(form "r")  
    ; is the filename valid?  
    if( isFile(form->fileName->value) then  
        hiSetCallbackStatus(form t); status is OK  
        hiFormUnmap(form); unmap the form  
; process filename...  
    else
```

Cadence User Interface SKILL Functions Reference

Forms

```
        ; keep the form on screen
        hiSetCallbackStatus(form nil)
        ; print out an error message
        printf("Invalid file '%s'.\n"
            form->fileName->value)
    )
)
```

Cadence User Interface SKILL Functions Reference

Forms

hiGetCurrentField

```
hiGetCurrentField(  
    r_form  
)  
=> s_fieldName / nil
```

Description

Returns the symbol name of the field that currently has input focus. The type of field that can be returned by this function is determined by the `?fieldFocus` setting on the form at the time of form instantiation.

For more information about `?fieldFocus`, see [hiCreateAppForm](#).

Arguments

<i>r_form</i>	Form handle. The form cannot be a scroll region field, but the current field may be within a scroll region field.
---------------	---

Value Returned

<i>s_fieldName</i>	Returns the field where the cursor is located.
<i>nil</i>	Returns <i>nil</i> if there are no type-in fields on the form.

Cadence User Interface SKILL Functions Reference

Forms

hiGetFieldInfo

```
hiGetFieldInfo(  
    r_2DFormOr2DMenu  
    s_field  
)  
=> l_dimensions / nil
```

Description

Returns the dimensions of any form field or 2D menu item that is in an instantiated form. Do not use the return value of `hiGetFieldInfo` to restore a field to its original location. Instead, use the values you specified originally.

`hiGetFieldInfo` does not work for uninstantiated forms. You can instantiate a form with `hiInstantiateForm` and then use `hiGetFieldInfo`.

Arguments

<i>r_2DFormOr2DMenu</i>	Form or menu returned from <code>hiCreateAppForm</code> , <code>hiCreateForm</code> , or <code>hiCreate2DMenu</code> .
<i>s_field</i>	SKILL symbol representing the field in <i>r_form</i> to be queried. This symbol is the field handle (?name) argument when the field was created.

Value Returned

<i>l_dimensions</i>	Returns a list containing the x, y, width, height, and promptBox width (if any) of the field. Note: Do not use the return value of <code>hiGetFieldInfo</code> to restore a field to its original location. Instead, use the values you specified originally.
<i>nil</i>	Returns <i>nil</i> and an error message if the list is not returned.

Example

```
list((x y) (width height) pBoxWidth )
```

hiGetFieldOverlaps

```
hiGetFieldOverlaps(  
    r_formHandle  
)  
=>t/nil
```

Description

Generates warnings if the form specified contains any fields whose bounding boxes overlap with other fields, or any fields in which the bounding box for the prompt portion of the field overlaps with the main portion of the field. The function generates an initial warning that overlaps were found, then additional warnings for each overlap. Returns `nil` if the form is uninstantiated; `t` otherwise.

This function is useful for checking form integrity while migrating from older releases to the 4.4.n release. It does not descend into scroll region fields and does not take a scroll region field as an argument.

Arguments

<i>r_formHandle</i>	Handle to the form which is to be checked for overlapping fields. <i>r_formHandle</i> cannot take a scroll region field as its value.
---------------------	--

Value Returned

<code>t</code>	Always returns <code>t</code> , except when the form is uninstantiated.
<code>nil</code>	Returns <code>nil</code> when the form is uninstantiated.

Cadence User Interface SKILL Functions Reference

Forms

hiGetCurrentForm

```
hiGetCurrentForm(  
    )  
=> r_form
```

Description

Returns the last active form, that is, the form that the cursor was last in.

Arguments

None.

Cadence User Interface SKILL Functions Reference

Forms

hiGetInsertionPosition

```
hiGetInsertionPosition(  
    g_form  
    s_field  
)  
=> x_position / nil
```

Description

Gets the current position of the insertion cursor in a type-in field. You can use the `hiSetInsertionPosition` function to change the position of the cursor.

Arguments

<i>g_form</i>	Handle to the form, which is returned by hiCreateAppForm .
<i>s_field</i>	Symbol name of the field.

Value Returned

<i>x_position</i>	An integer that represents the position of the insertion cursor. The beginning of the field is 0, the first character is 1, and so on.
nil	The position of the insertion cursor could not be obtained.

Reference

[hiSetInsertionPosition](#)

Cadence User Interface SKILL Functions Reference

Forms

hiGetLayerCyclicValue

```
hiGetLayerCyclicValue(  
    d_techFileId  
    r_form  
)  
=> d_layerId / nil
```

Description

Returns the layer object associated with the current value of the layer cyclic field.

This function should be called only if the form contains a layer cyclic field (see [hiCreateLayerCyclicField](#)).

Arguments

<i>d_techFileId</i>	The techfile object associated with the layers of the cyclic field.
<i>r_form</i>	Form containing the layer cyclic field. This form handle is returned from hiCreateAppForm .

Value Returned

<i>d_layerId</i>	Returns the layer (database object) on successful completion.
<i>nil</i>	Returns <i>nil</i> and an error message if an error occurs.

Cadence User Interface SKILL Functions Reference

Forms

hiGetListBoxValue

```
hiGetListBoxValue(  
    o_listBox  
)  
=> l_stringValues
```

Description

Provides the value of a list box field as a list of strings. This function is useful when the `valueByPosition` property of a list box field is set to `non-nil`, which means its value is managed and reported as a list of integers (indicating position) instead of a list of strings.

Note that the return value of this function does not retain positional information, so if the list box field has multiple identical strings that you identify by their position, the return value will not provide that information.

See the description of the `?valueByPosition` argument of [hiCreateListBoxField](#) for more information.

Arguments

<code>o_listBox</code>	Handle to the list box field.
------------------------	-------------------------------

Value Returned

<code>l_stringValues</code>	Value of the list box field as a list of strings.
-----------------------------	---

Reference

[hiCreateListBoxField](#)

Cadence User Interface SKILL Functions Reference

Forms

hiGetScrollBarInfo

```
hiGetScrollBarInfo(  
    g_formOrScrollRegion  
)  
=> l_scrollbarInfo
```

Description

Provides information about the horizontal and vertical scroll bars on a form or scroll region field. You can use this information to set the position of scroll bars with the `hiSetScrollBarValue` function.

Arguments

g_formOrScrollRegion
Handle to the form or scroll region field.

Value Returned

l_scrollbarInfo A list of the following form:

```
list( [nil | list( x_horzMin x_horzMax x_horzValue x_horzSliderSize)]  
      [nil | list( x_vertMin x_vertMax x_vertValue x_vertSliderSize)] )
```

The first element of the list provides information about the horizontal scroll bar; the second element of the list provides information about the vertical scroll bar. If the first element is `nil`, the form or scroll region field does not have a horizontal scroll bar; if the second item in the list is `nil`, the form or scroll region field does not have a vertical scroll bar.

x_horzMin and *x_vertMin* are the minimum pixel position (usually 0); *x_horzMax* and *x_vertMax* are the horizontal or vertical pixel size of the form or scroll region field; *x_horzValue* and *x_vertValue* are the current values of the scroll bars; and *x_horzSliderSize* and *x_vertSliderSize* are the visible areas of the form or scroll region field.

Note: The maximum values for the *l_newValues* argument of the `hiSetScrollBarValue` function are

Cadence User Interface SKILL Functions Reference

Forms

x_horzMax - x_horzSliderSize and
x_vertMax - x_vertSliderSize.

Reference

[hiSetScrollBarValue](#)

Cadence User Interface SKILL Functions Reference

Forms

hiGetTextFieldFit

```
hiGetTextFieldFit(  
    x_rows  
    x_columns  
    t_font  
    g_hasVerticalScrollbar  
    g_hasHorizontalScrollbar  
)  
=> l_dimensions
```

Description

Computes and returns the pixel size needed for a multiline text field entry for a form. The value returned is based on the specified font and character extents.

For more information about multiline text fields, see [hiCreateMLTextField](#).

Arguments

<i>x_rows</i>	The number of rows to display assuming a horizontal scroll bar may be present.
<i>x_columns</i>	The number of columns to display assuming a vertical scroll bar may be present.
<i>t_font</i>	The font that will be used in the text field. It must be set to <code>nil</code> , causing the font to be derived from the defaults. The default is determined by the user's <code>.cdsinit</code> file. If no font is specified in <code>.cdsinit</code> , the " <i>fixed</i> " font will be used. Specifying a specific font name for this argument is not supported.
<i>g_hasVerticalScrollbar</i> (t/nil)	Indicates if there should be a vertical scroll bar displayed for the field.
<i>g_hasHorizontalScrollbar</i> (t/nil)	Indicates if there should be a horizontal scroll bar displayed for the field.

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

<i>l_dimensions</i>	Returns a list containing the width and height of the theoretical text field on successful completion.
<i>nil</i>	Return <i>nil</i> if an error occurs.

Cadence User Interface SKILL Functions Reference

Forms

hiGetTopListItem

```
hiGetTopListItem(  
    o_listbox  
)  
=> x_position / nil
```

Description

Returns the top item position of a listbox field.

Arguments

<i>o_listbox</i>	SKILL handle to the listbox.
------------------	------------------------------

Value Returned

<i>x_position</i>	The top item position of the listbox field. 1 is the first item.
<i>nil</i>	Return <i>nil</i> if the listbox is invalid or if the form hasn't been instantiated.

hiHighlightField

```
hiHighlightField(  
    g_form  
    s_field  
    g_highlightType  
)  
=> t / nil
```

Description

Highlights a type-in field depending on the value of *g_highlightType*.

Highlighting is allowed for type-in fields only, as these are the fields into which invalid data might be entered, or data may be required.

To find the field, this function also searches the scroll region fields within the form or scroll region field specified as *r_form*. It does a depth-first hierarchical search, which means that scroll region fields within a form or scroll region field are searched before the top level of the form or scroll region field.

Arguments

<i>g_form</i>	Form (returned from hiCreateAppForm or hiCreateForm) or scroll region field (returned from hiCreateScrollRegion). Scroll region fields within the form or scroll region field are also searched.
<i>s_field</i>	SKILL field handle representing the field you wish to highlight. The field must be a text-entry (type-in) field only.
<i>g_highlightType</i>	<p>It can be either a symbol or an integer. As a symbol, it must be <code>error</code>, <code>highlight</code> or <code>background</code> to set the background color to an error, highlight, or background (default) color, respectively. As an integer, it must be <code>hicError</code>, <code>hicHighlight</code>, or <code>hicBackground</code>.</p> <p>If invalid data has been entered, error color should be specified. highlight color may be used to indicate required data. To reset the color to the original background color, background color should be used.</p>

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

<code>t</code>	Returns <code>t</code> if highlighting is set.
<code>nil</code>	Returns <code>nil</code> and an error message if the highlighting is not set.

Cadence User Interface SKILL Functions Reference

Forms

hiIgnoreProp

```
hiIgnoreProp(  
    s_objType  
    t_propName  
)  
=> t / nil
```

Description

Does not display the specified property of a given object type in the Property List Editor form.

This function can be called multiple times on the same object type so that multiple properties can be ignored.

Arguments

<i>s_objType</i>	Symbol corresponding to the database object's <i>objType</i> attribute. (For example: <i>cellView</i> , <i>inst</i> , <i>pin</i> , <i>lib</i> , <i>version</i> .)
<i>t_propName</i>	Name of the property to ignore. If this property has been stored on an object of the specified <i>s_objType</i> , it is not displayed.

Note: Object attributes cannot be ignored.

Value Returned

<i>t</i>	Returns <i>t</i> if the property is not displayed.
<i>nil</i>	Issues a warning and returns <i>nil</i> if the object type or property name is not valid.

Example

The properties *privateProp* and *ignoreMe* are not displayed when displaying the Property List Editor form for a database *cellview*.

```
hiIgnoreProp( 'cellView "privateProp" )  
hiIgnoreProp( 'cellView "ignoreMe")  
hiEditPropList( cell )
```

Cadence User Interface SKILL Functions Reference

Forms

hiInFormApply

```
hiInFormApply(  
    r_form  
)  
=> t / nil
```

Description

Checks whether the `Apply` button on a mapped form is selected.

Arguments

<i>r_form</i>	Name of the form.
---------------	-------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the <code>Apply</code> button on a mapped form is selected.
----------	---

<i>nil</i>	Returns <i>nil</i> if the <code>Apply</code> button is not selected or the form is not mapped.
------------	--

Cadence User Interface SKILL Functions Reference

Forms

hiInstantiateForm

```
hiInstantiateForm(  
    r_form  
)  
=> t / nil
```

Description

Instantiates the C structure of a form from SKILL.

If a form is uninstantiated, field callbacks do not get called, since the form is simply a SKILL structure at this point. Instantiating the form gives you full form functionality, but the form will not be displayed.

Arguments

<i>r_form</i>	A form returned by hiCreateAppForm (preferably) or hiCreateForm .
---------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the form is instantiated.
<i>nil</i>	Returns <i>nil</i> and an error message if an error occurs.

Example

This example instantiates the form `myform`.

```
hiInstantiateForm( myform) => t
```

Cadence User Interface SKILL Functions Reference

Forms

hIsForm

```
hIsForm(  
    g_FormOrMenu  
)  
=> t / nil
```

Description

Checks whether a form or menu is a valid one.

Arguments

<i>g_FormOrMenu</i>	Form or menu whose validity is being checked.
---------------------	---

Value Returned

t	Returns t if the form or menu is valid.
---	---

nil	Returns nil if the form or menu is not valid.
-----	---

Cadence User Interface SKILL Functions Reference

Forms

hiIsFormDisplayed

```
hiIsFormDisplayed(  
    r_form  
)  
=> t / nil
```

Description

Checks whether a form is displayed.

Arguments

<i>r_form</i>	Form being checked for visibility.
---------------	------------------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the form is displayed.
----------	--

<i>nil</i>	Returns <i>nil</i> if the form is not displayed or is not instantiated.
------------	---

Cadence User Interface SKILL Functions Reference

Forms

hiIsInstantiated

```
hiIsInstantiated(  
    r_formOrMenu  
)  
=> t / nil
```

Description

Returns *t* if a form has been displayed at least once or a menu has been placed in a window.

Arguments

<i>r_formOrMenu</i>	Form or menu whose validity is being checked.
---------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the form or menu is valid.
<i>nil</i>	Returns <i>nil</i> if the form or menu is not valid.

Cadence User Interface SKILL Functions Reference

Forms

hiIsInFieldCancel

```
hiIsInFieldCancel(  
    )  
=> r_formHandle/nil
```

Description

Query this function during a field or form callback to determine if the form is currently being cancelled (the callback may be initiated by the cancel operation). If the form is being cancelled, the form handle is returned; otherwise `nil` is returned.

Arguments

This function does not take any arguments.

Value Returned

<i>r_formHandle</i>	Returns the handle to the form if the form is currently being cancelled.
<code>nil</code>	Returns <code>nil</code> if the form is not being cancelled.

Cadence User Interface SKILL Functions Reference

Forms

hiLayerMatchCyclicStr

```
hiLayerMatchCyclicStr(  
    l_choices  
    t_layer  
)  
=> l_layerValue
```

Description

Returns the layer cyclic value associated with the specified *t_layer*.

This is a convenience function that is used to search the specified list of iconic layer choices (*l_choices*) for a value matching the *t_layer* string. This function should only be used for layer cyclic fields (see [hiCreateLayerCyclicField](#)).

Arguments

<i>l_choices</i>	List of iconic layer lists created from a call to <code>hiCreateLayerCyclicField</code> . This list can be retrieved from <code>form->hiLayerField->choices</code> .
<i>t_layer</i>	String representing the layer you wish to match. This string is in the form of <i>t_layerName</i> (<i>t_layerPurpose</i>). (This format is equivalent to the fourth element of a layer iconic value.)

Value Returned

<i>l_layerValue</i>	Returns the cyclic value of the specified layer.
---------------------	--

Cadence User Interface SKILL Functions Reference

Forms

hiLayerStringToLPP

```
hiLayerStringToLPP(  
    d_techFileId  
    t_layer  
)  
=> d_layerId / nil
```

Description

Returns the layer object in the specified *d_techFileId* associated with the specified *t_layer*.

This function is used in conjunction with layer cyclic fields (see [hiCreateLayerCyclicField](#)).

Arguments

<i>d_techFileId</i>	The techfile object associated with the layers of the cyclic field.
<i>t_layer</i>	String representing the layer you wish to convert. This string is in the form of <i>t_layerName</i> (<i>t_layerPurpose</i>). (This format is equivalent to the fourth element of a layer iconic value.)

Value Returned

<i>d_layerId</i>	Returns the corresponding database layer object.
<i>nil</i>	Returns <i>nil</i> if the layer is not found in the specified techfile.

Cadence User Interface SKILL Functions Reference

Forms

hiMoveField

```
hiMoveField(  
    r_2DFormOr2DMenu  
    s_field  
    l_location  
)  
=> t / nil
```

Description

Moves the specified field to a new location. This function cannot be used for fields that have field attachments. Use `hiReattachField` for fields with field attachments.

Arguments

<i>r_2DFormOr2DMenu</i>	2D form or menu returned from hiCreateAppForm , hiCreateForm , or hiCreate2DMenu .
<i>s_field</i>	SKILL symbol representing the field in <i>r_form</i> you wish to move.
<i>l_location</i>	<p>Location to which you want to move the field. The location is relative to the upper-left corner of the form. <i>location</i> is of the form:</p> <pre>list(x_xLoc x_yLoc) or x_xLoc : y_yLoc</pre> <p>The location is relative to the upper-left corner of the form.</p>

Value Returned

<i>t</i>	Returns <i>t</i> if the field is moved.
<i>nil</i>	Returns <i>nil</i> and an error message if the field is not moved.

hiMoveInsBarToEnd

```
hiMoveInsBarToEnd(  
    r_form  
    s_field  
)  
=> t / nil
```

Description

Places the cursor (insertion point) at the end of a text field, scrolling the field so the cursor is visible.

This function does *not* make the field active if it is not already so.

The field must be instantiated and must be the current field (must have input focus).

Arguments

<i>r_form</i>	Form containing the field where the cursor is to be placed.
<i>s_field</i>	Field where the cursor is placed. The field must be a type-in field (string, float, integer, list, etc.) and must exist in the specified form.

Value Returned

<i>t</i>	The cursor was placed at the end of the field.
<i>nil</i>	The field is not on the form or is not a type-in field. An error message also prints.

Example

```
hiMoveInsBarToEnd( myform `myTypeinField) => t
```

Cadence User Interface SKILL Functions Reference

Forms

hiOffsetField

```
hiOffsetField(  
    r_2DFormOr2DMenu  
    s_field  
    l_offsets  
)  
=> t / nil
```

Description

Offsets one field from its current position. This function cannot be used for fields that have field attachments. Use `hiReattachField` for fields with field attachments.

Note: This routine adds the x and y offset values and the current x and y values to determine the new field position. An offset of 0 has no effect. Negative offsets may be specified, but the minimum value of the new x and y is 0. Both offset values are required.

Use this function to offset a single field. To offset more than one field, use [`hiOffsetFields`](#).

The form must be instantiated.

Arguments

<i>r_2DFormOr2DMenu</i>	Two-dimensional form or menu returned from <code>hiCreateAppForm</code> , <code>hiCreateForm</code> , or <code>hiCreate2DMenu</code> .
<i>s_field</i>	SKILL symbol representing the field in <i>r_form</i> you wish to offset.
<i>l_offsets</i>	Distance in pixels the field is set off from the current position. <i>offsets</i> is of the form: <pre>list(x_xOffset x_yOffset) or x_xOffset : x_yOffset</pre>

Value Returned

<i>t</i>	Returns <i>t</i> if the field is offset.
<i>nil</i>	Returns <i>nil</i> and an error message if the field is not offset.

Cadence User Interface SKILL Functions Reference

Forms

Example

```
hiOffsetField( testform `myField 10:20) => t
```

hiOffsetFields

```
hiOffsetFields(  
    r_2DFormOr2DMenu  
    l_fields  
    l_offsets  
)  
=> t / nil
```

Description

Offsets several fields from their current positions. All fields are offset by the same distance (*l_offsets*); they move as a block. This function cannot be used for fields that have field attachments. Use `hiReattachField` for fields with field attachments.

This routine adds the x and y offset values and the current x and y values to determine the new field position. An offset of 0 will have no effect. Negative offsets can be specified, but the minimum value of the new x and y is 0. Both offset values are required.

The form must be instantiated.

To offset only one field, use `hiOffsetField`.

Arguments

<i>r_2DFormOr2DMenu</i>	Two-dimensional form or menu returned from <code>hiCreateAppForm</code> , <code>hiCreateForm</code> , or <code>hiCreate2DMenu</code> .
<i>l_fields</i>	List representing the fields in <i>r_form</i> you wish to offset.
<i>l_offsets</i>	Distance in pixels the field is set off from the current position. <i>offsets</i> is of the form: <pre>list(x_xOffset x_yOffset) or x_xOffset : x_yOffset</pre>

Value Returned

t	Returns t if the fields are offset.
nil	Returns nil and an error message if the fields are not offset.

Cadence User Interface SKILL Functions Reference

Forms

Example

```
hiOffsetFields( testform `(myField myField2) -10:0) => t
```


hiResizeField

```
hiResizeField(  
    r_2DForm  
    s_field  
    l_resizeDescription  
)  
=> t / nil
```

Description

Changes the width, height, or *promptBox* width of a specified field. This function cannot be used for fields that have field attachments. Use `hiReattachField` for fields with field attachments.

Arguments

r_2DForm 2D form returned from `hiCreateAppForm` or `hiCreateForm`.

s_field SKILL field handle representing the field you wish to resize.

l_resizeDescription New dimensions, in the following form:

```
list( x_width x_height [x_promptBoxWidth])
```

x_width and *x_height*: Required. The new width and height of the field.

x_promptBoxWidth: Optional, and is considered only if a prompt exists. If this value is not specified for a field which has a prompt, the prompt box for the field does not change.

Note: For separator fields the height is ignored and the width as the length of the separator whether it is vertical or horizontal.

Value Returned

t Returns *t* if the field characteristics are changed.

nil Returns *nil* and an error message if the field characteristics are not changed.

hiReattachField

```
hiReattachField(  
    g_formOrScrollRegion  
    s_field  
    l_positionalInfo  
    i_fieldAttachment  
)
```

Description

Reattaches a field in a two-dimensional form or scroll region that has field attachments. This function places the field according to the new position and attachment information. This function can be used only with forms or scroll regions that already have field attachments. For forms or scroll regions that do not have field attachments, use the `hiMoveField`, `hiResizeField`, `hiOffsetField`, and `hiOffsetFields` functions.

Arguments

g_formOrScrollRegion

Handle to the two-dimensional form or scroll region that contains the field you want to reattach.

s_field

Handle to the field you want to reattach.

i_positionalInfo

The two-dimensional attributes for the field. Use the following syntax:

```
list( x:y width:height [promptBoxWidth])
```

For details about two-dimensional attributes, see [“Two Dimensional Form Layout”](#) on page 212.

i_fieldAttachment

A bitwise OR of a list of constants. The field attachment determines how the two-dimensional attributes of a field are used to position the field in the form. This argument is valid only for two-dimensional forms. For a list of constants and details about how to use field attachments, see [“Field Attachments”](#) on page 214.

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

<code>t</code>	Returns <code>t</code> if the field characteristics are changed.
<code>nil</code>	Returns <code>nil</code> and an error message if the field characteristics are not changed.

Cadence User Interface SKILL Functions Reference

Forms

hiReportSelectItem

```
hiReportSelectItem(  
    g_reportField  
    g_itemIndex  
    [g_notify]  
)  
=> t/nil
```

Description

Selects an item in a report field. This function also executes the callback of the report field if you specify *t* as the value of the *g_notify* argument.

This function is the programmatic equivalent of selecting an item with the mouse or keyboard keys.

To select more than one item, use [hiReportSelectItems](#); to select all items in a field, use [hiReportSelectAllItems](#).

Arguments

<i>g_reportField</i>	The report field that contains the item.
<i>g_itemIndex</i>	The item you want to select, specified as an integer indicating the item's position in <i>?choices</i> . The first item in <i>?choices</i> is 0, the second is 1, and so on. The <i>?choices</i> argument is specified when the field is created with hiCreateReportField .
<i>g_notify</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want to trigger the callback procedure of the report field. Specify <i>nil</i> if you do not want to trigger the callback procedure. The default value is <i>nil</i> . For more information about the callback, see hiCreateReportField .

Value Returned

<i>t</i>	The item was selected.
<i>nil</i>	The item was not selected.

Cadence User Interface SKILL Functions Reference

Forms

Reference

[hiReportSelectItems](#), [hiReportSelectAllItems](#), [hiReportDeselectItem](#),
[hiReportDeselectItems](#), [hiReportDeselectAllItems](#), [hiReportGetSelectedItems](#),
[hiCreateReportField](#)

Cadence User Interface SKILL Functions Reference

Forms

hiReportSelectItems

```
hiReportSelectItems(  
    g_reportField  
    l_itemIndexList  
    [g_notify]  
)  
=> t/nil
```

Description

Selects items in a report field. This function also executes the callback of the report field if you specify `t` as the value of the `g_notify` argument.

This function is the programmatic equivalent of selecting items with the mouse or keyboard keys.

To select a single item, use [hiReportSelectItem](#); to select all items in a field, use [hiReportSelectAllItems](#).

Arguments

<code>g_reportField</code>	The report field that contains the items.
<code>l_itemIndexList</code>	The items you want to select, specified as a list of integers indicating the items' positions in <code>?choices</code> . The first item in <code>?choices</code> is 0, the second is 1, and so on. You can specify a range of items as a list within the list.

For example:

```
'(1 3 6)           ; selects items 1, 3, 6  
'('(2 4))         ; selects items 2, 3, 4  
'('(1 3) 9 11)    ; selects items 1, 2, 3, 9, 11
```

The `?choices` argument is specified when the field is created with [hiCreateReportField](#).

<code>g_notify</code>	<code>t</code> or <code>nil</code> . Specify <code>t</code> if you want to trigger the callback procedure of the report field. Specify <code>nil</code> if you do not want to trigger the callback procedure.
-----------------------	---

The default value is `nil`.

Cadence User Interface SKILL Functions Reference

Forms

For more information about the callback, see [hiCreateReportField](#).

Value Returned

t	The items were selected.
nil	The items were not selected.

Reference

[hiReportSelectItem](#), [hiReportSelectAllItems](#), [hiReportDeselectItem](#),
[hiReportDeselectItems](#), [hiReportDeselectAllItems](#), [hiReportGetSelectedItems](#),
[hiCreateReportField](#)

hiReportSelectAllItems

```
hiReportSelectAllItems(  
    g_reportField  
    [g_notify]  
)  
=> t/nil
```

Description

Selects all items in a report field. This function also executes the callback of the report field if you specify *t* as the value of the *g_notify* argument.

This function is the programmatic equivalent of selecting items with the mouse or keyboard keys.

To select a single item, use [hiReportSelectItem](#); to select more than one item, use [hiReportSelectItems](#).

Arguments

<i>g_reportField</i>	The report field that contains the items.
<i>g_notify</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want to trigger the callback procedure of the report field. Specify <i>nil</i> if you do not want to trigger the callback procedure. The default value is <i>nil</i> . For more information about the callback, see hiCreateReportField .

Value Returned

<i>t</i>	The items were selected.
<i>nil</i>	The items were not selected.

Reference

[hiReportSelectItem](#), [hiReportSelectItems](#), [hiReportDeselectItem](#), [hiReportDeselectItems](#), [hiReportDeselectAllItems](#), [hiReportGetSelectedItems](#), [hiCreateReportField](#)

hiReportDeselectItem

```
hiReportDeselectItem(  
    g_reportField  
    g_itemIndex  
    [g_notify]  
)  
=> t/nil
```

Description

Deselects an item in a report field. This function also executes the callback of the report field if you specify *t* as the value of the *g_notify* argument.

To deselect more than one item, use [hiReportDeselectItems](#); to deselect all selected items in a field, use [hiReportDeselectAllItems](#).

This function is the programmatic equivalent of deselecting items with the mouse or keyboard keys.

Arguments

<i>g_reportField</i>	The report field that contains the item.
<i>g_itemIndex</i>	The item you want to deselect, specified as an integer indicating the item's position in <i>?choices</i> . The first item in <i>?choices</i> is 0, the second is 1, and so on. The <i>?choices</i> argument is specified when the field is created with hiCreateReportField .
<i>g_notify</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want to trigger the callback procedure of the report field. Specify <i>nil</i> if you do not want to trigger the callback procedure. The default value is <i>nil</i> . For more information about the callback, see hiCreateReportField .

Value Returned

<i>t</i>	The item was deselected
<i>nil</i>	The item was not deselected.

Cadence User Interface SKILL Functions Reference

Forms

Reference

[hiReportSelectItem](#), [hiReportSelectItems](#), [hiReportSelectAllItems](#),
[hiReportDeselectItems](#), [hiReportDeselectAllItems](#), [hiReportGetSelectedItems](#),
[hiCreateReportField](#)

hiReportDeselectItems

```
hiReportDeselectItems(  
    g_reportField  
    l_itemIndexList  
    [g_notify]  
)  
=> t/nil
```

Description

Deselects items in a report field. This function also executes the callback of the report field if you specify `t` as the value of the `g_notify` argument.

To deselect a single item, use [hiReportDeselectItem](#); to deselect all items in a field, use [hiReportDeselectAllItems](#).

This function is the programmatic equivalent of deselecting items with the mouse or keyboard keys.

Arguments

g_reportField The report field that contains the items.

l_itemIndexList The items you want to deselect, specified as a list of integers indicating the items' positions in `?choices`. The first item in `?choices` is 0, the second is 1, and so on. You can specify a range of items as a list within the list.

For example:

```
'(1 3 6)                              ; deselects items 1, 3, 6  
'('(2 4))                             ; deselects items 2, 3, 4  
'('(1 3) 9 11)                       ; deselects items 1, 2, 3, 9, 11
```

The `?choices` argument is specified when the field is created with [hiCreateReportField](#).

g_notify `t` or `nil`. Specify `t` if you want to trigger the callback procedure of the report field. Specify `nil` if you do not want to trigger the callback procedure.

The default value is `nil`.

Cadence User Interface SKILL Functions Reference

Forms

For more information about the callback, see [hiCreateReportField](#).

Value Returned

t	The items were deselected.
nil	The items were not deselected.

Reference

[hiReportSelectItem](#), [hiReportSelectItems](#), [hiReportSelectAllItems](#),
[hiReportDeselectItem](#), [hiReportDeselectAllItems](#), [hiReportGetSelectedItems](#),
[hiCreateReportField](#)

hiReportDeselectAllItems

```
hiReportDeselectAllItems(  
    g_reportField  
    [g_notify]  
)  
=> t/nil
```

Description

Deselects all selected items in a report field. This function also executes the callback of the report field if you specify `t` as the value of the `g_notify` argument.

To deselect a single item, use [hiReportDeselectItem](#); to deselect more than one item, use [hiReportDeselectItems](#).

This function is the programmatic equivalent of deselecting items with the mouse or keyboard keys.

Arguments

<code>g_reportField</code>	The report field.
<code>g_notify</code>	<code>t</code> or <code>nil</code> . Specify <code>t</code> if you want to trigger the callback procedure of the report field. Specify <code>nil</code> if you do not want to trigger the callback procedure. The default value is <code>nil</code> . For more information about the callback, see hiCreateReportField .

Value Returned

<code>t</code>	The items were deselected.
<code>nil</code>	The items were not deselected.

Reference

[hiReportSelectItem](#), [hiReportSelectItems](#), [hiReportSelectAllItems](#), [hiReportDeselectItem](#), [hiReportDeselectItems](#), [hiReportGetSelectedItems](#), [hiCreateReportField](#)

Cadence User Interface SKILL Functions Reference

Forms

hiReportGetSelectedItems

```
hiReportGetSelectedItems(  
    g_reportField  
)  
=> l_selectedItemsIndex / nil
```

Description

Gets a list of all the items that are currently selected in a report field.

Arguments

g_reportField The report field.

Value Returned

l_selectedItemsIndex

List of integers representing the items that are currently selected in the report field. The integers represent the positions of the selected items in ?choices. The first item in ?choices is 0, the second is 1, and so on. The ?choices argument is specified when the field is created with [hiCreateReportField](#).

nil There was an error.

Reference

[hiReportSelectItem](#), [hiReportSelectItems](#), [hiReportSelectAllItems](#),
[hiReportDeselectItem](#), [hiReportDeselectItems](#), [hiReportDeselectAllItems](#),
[hiCreateReportField](#)

hiSetButtonLabel

```
hiSetButtonLabel(  
    g_Button  
    g_Label  
)  
=> t / nil
```

Description

Changes a button's label text or icon after the button is created. This function updates all instances of the button. You can also set the button text or icon directly with the `buttonText` and `buttonIcon` properties.

To set the button text or icon when the button is not instantiated, use the following properties:

- `button->buttonText = "value"`
- `button->buttonIcon = value`

To set the button text or icon after the button is instantiated, use the following properties:

- `form->button->buttonText = "value"`
- `form->button->buttonIcon = value`

Note: `hiSetButtonLabel` does not change an instance's original size.

Arguments

<i>g_Button</i>	The handle to the button returned by <code>hiCreateButton</code> .
<i>g_Label</i>	The new label text (string) or icon.

Value Returned

<i>t</i>	Returns <i>t</i> if the button label is changed.
<i>nil</i>	Returns <i>nil</i> if the button label was not changed.

Cadence User Interface SKILL Functions Reference

Forms

hiSetCallbackStatus

```
hiSetCallbackStatus(  
    r_form  
    g_booleanStatus  
)  
=> t / nil
```

Description

Overrides the removal (unmapping) of a form when the form's *OK* button is pressed.

This function can be used in the form's callback and is useful if you enter invalid form data and the application instructs you to re-enter a response.

Note: To use this function, you must set the *unmapAfterCB* flag argument to *hiCreateAppForm* to *t*.

Arguments

<i>r_form</i>	Form description returned from hiCreateAppForm or <i>hiCreateForm</i> .
<i>g_booleanStatus</i>	Desired status of the form, <i>t</i> or <i>nil</i> . This <i>booleanStatus</i> is a toggle value, and must be reset to <i>t</i> once it has been set to <i>nil</i> . The form will not be brought down until this value is reset (see below). This routine should only be used with standard forms; the user should never try to override the actions of an options form. Each form will have its initial status set (default is <i>t</i>) to undisplay the form when <i>OK</i> is pressed. This value is reset to <i>t</i> if the user cancels the form.

Value Returned

<i>t</i>	Returns <i>t</i> on successful completion.
<i>nil</i>	Returns <i>nil</i> and an error message if an error occurs.

hiSetCurrentField

```
hiSetCurrentField(  
    g_form  
    s_fieldName  
    [g_limitSearch]  
)  
=> t / nil
```

Description

Makes the specified type-in field the current field (sets input focus to the field). If the field is not visible, *hiSetCurrentField* scrolls the form or scroll region field so the field becomes visible.

The fields that accept focus are determined by the *?fieldFocus* argument of the *hiCreateAppForm* function.

The function first searches for the field at the top level of the form or scroll region field specified. If the field is not found at the top level, all scroll region fields within the form or scroll region field are searched hierarchically. The first field that matches *s_fieldName* is set as the current field. You can limit the search to only the top level of the form or scroll region field by specifying a non-*nil* value for the third argument, *limitSearch*.

Arguments

<i>g_form</i>	The form or scroll region field in which you want to set the current field.
<i>s_fieldName</i>	Field to be made current. If the field is not an editable text field, it is scrolled into view but not given focus. If the field is not found at the top-level of the form or scroll region field, all scroll region fields within the form or scroll region field are searched hierarchically. The first match is selected.
<i>g_limitSearch</i>	Any non- <i>nil</i> value. Use this argument if you want to limit the search for the field to only the top level of the form or scroll region field specified. If this argument is not used, the function first searches the top level of the form or scroll region field and then, if the field is not found, does a hierarchical search of all scroll region fields within the form or scroll region field.

Cadence User Interface SKILL Functions Reference

Forms

Value Returned

<code>t</code>	The field was made current (input focus was set to the field).
<code>nil</code>	Input focus could not be set to the field.

Cadence User Interface SKILL Functions Reference

Forms

hiSetFieldEditable

```
hiSetFieldEditable(  
    r_field  
    g_editable  
)  
=> t / nil
```

Description

Sets the specified field editable or noneditable, in all forms where it appears.

This function should be used after the form containing the field has been created. You can also make a field editable or noneditable using:

```
form->field->editable=t/nil
```

Arguments

r_field Field must be a text-entry (type-in) field returned from a field creation routine. If the field is already placed in a form, the field must be specified through the form handle, that is,

```
form->field
```

g_editable If set to `nil`, the field is read-only; setting *editable* to `t` makes the field editable. Read-only fields can accept focus through tabbing or clicking the mouse only if the *?fieldFocus* argument of `hiCreateAppForm` is set to `'allTypein`. For more information, see [hiCreateAppForm](#). On eight-plane displays, the text entry area of non-editable fields is greyed out and the field prompt is stippled.

Value Returned

`t` Returns `t` if the editable flag is set.

`nil` Returns `nil` and an error message if the editable flag is not set.

Cadence User Interface SKILL Functions Reference

Forms

hiSetFieldEnabled

```
hiSetFieldEnabled(  
    g_field  
    g_enabled  
)  
=> t / nil
```

Description

Enables or disables a form field. A disabled field is greyed-out. This function updates all instances of the field. You can use `hiSetFieldEnabled` with all types of fields except scroll region fields and separator fields.

You can also enable or disable a field with the following property:

```
form->field->enabled = t / nil
```

To enable or disable buttons on a form, use [`hiSetFormButtonEnabled`](#).

Arguments

<i>g_field</i>	Handle to field returned from functions that create fields.
<i>g_enabled</i>	<code>t</code> or <code>nil</code> . If you specify <code>t</code> , the field is enabled; if you specify <code>nil</code> , the field is disabled, that is, greyed-out.

Value Returned

<code>t</code>	Returns <code>t</code> if the field was changed according to your specification.
<code>nil</code>	Returns <code>nil</code> if the field was not changed.

hiSetFormButtonEnabled

```
hiSetFormButtonEnabled(  
    r_form  
    s_buttonSym  
    g_enabled  
)  
=> t / nil
```

Description

Enables or disables a button on a form. To enable or disable fields on a form, use the `hiSetFieldEnabled` function.

Arguments

<i>r_form</i>	Form handle returned by <code>hiCreateAppForm</code> .
<i>s_buttonSym</i>	Button symbol.
<i>g_enabled</i>	<i>t</i> or <i>nil</i> . If you specify <i>t</i> , the button is enabled; if you specify <i>nil</i> , the button is disabled, that is, greyed-out.

Value Returned

<i>t</i>	Returns <i>t</i> if successful.
<i>nil</i>	Returns <i>nil</i> if there was an error.

Examples

```
hiSetFormButtonEnabled( myForm 'Apply nil)  
hiSetFormButtonEnabled( myForm 'Prev nil)  
hiSetFormButtonEnabled( myForm 'Help nil)
```

hiSetFormBlock

```
hiSetFormBlock(  
    r_form  
    [g_blockUpdates]  
    [g_unmanage]  
)  
=> t / nil
```

Description

Prevents time-consuming multiple updates to a form during multiple changes (such as adding, deleting, moving, or resizing elements). You can also blank out the form during extensive changes. Updates are blocked until you call `hiSetFormBlock(r_form nil)` or `hiUpdateFormBlock(r_form)`.

Note: Use the `hiSetFormBlock` function only if you are making extensive changes to a form.

Arguments

<i>r_form</i>	Handle to the form, returned by hiCreateAppForm .
<i>g_blockUpdates</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> to block updates to the form. Specify <i>nil</i> to reset the form to allow updates. You can also reset the form with the hiUpdateFormBlock function.
<i>g_unmanage</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> to blank out the form during extensive changes. Specify <i>nil</i> when you reset the form to allow updates. You can also reset the form with the hiUpdateFormBlock function.

Value Returned

<i>t</i>	Returns <i>t</i> unless <i>r_form</i> is not valid.
<i>nil</i>	Returns <i>nil</i> if <i>r_form</i> is not valid.

Example

```
hiSetFormBlock( r_form t )  
<add/delete/move/resize functions>  
hiSetFormBlock( r_form nil )
```

hiSetFormHighlights

```
hiSetFormHighlights(  
    g_form  
    g_highlightType  
)  
=> t / nil
```

Description

Highlights all type-in fields for *form*, depending on the value of *highlightType*.

Highlighting is allowed for type-in fields only, as these are the fields into which invalid data might be entered, or data may be required.

To find type-in fields, the function searches the form or scroll region field specified as *r_form*. It also does a hierarchical search of all scroll region fields within the form or scroll region field.

Arguments

<i>g_form</i>	Form or scroll region field.
<i>g_highlightType</i>	It can be either a symbol or an integer. As a symbol, it must be <code>error</code> , <code>highlight</code> , or <code>background</code> to set the background color to an error, highlight, or background (default) color, respectively. As an integer, it must be <code>hicError</code> , <code>hicHighlight</code> , or <code>hicBackground</code> .

Value Returned

<i>t</i>	Returns <i>t</i> if the highlighting flag is set.
<i>nil</i>	Returns <i>nil</i> and an error message if the highlighting flag is not set.

Cadence User Interface SKILL Functions Reference

Forms

hiSetFormMinMaxSize

```
hiSetFormMinMaxSize(  
    r_form  
    g_minSize  
    g_maxSize  
)  
=> t / nil
```

Description

Sets the minimum or maximum size of the form. The size excludes the window manager's title and border.

Arguments

<i>r_form</i>	Form whose minimum or maximum size you want to set.
<i>g_minSize</i>	Minimum size of the form, specified as a list of width and height or <i>width:height</i> ; or <i>nil</i> .
<i>g_maxSize</i>	Maximum size of the form, specified as a list of width and height or <i>width:height</i> ; or <i>nil</i> .

Value Returned

<i>t</i>	Returns <i>t</i> if successful.
<i>nil</i>	Returns <i>nil</i> if there was an error.

hiSetFormName

```
hiSetFormName(  
    r_form  
    t_newTitle  
)  
=> t
```

Description

Changes the form's title (which is displayed in the window frame banner). The function can be called by its alias `hiChangeFormTitle`.

Arguments

<i>r_form</i>	Form whose title is being changed.
<i>t_newTitle</i>	New title.

Value Returned

<i>t</i>	Returns <i>t</i> after the title is changed.
----------	--

Reference

[hiChangeFormTitle](#)

Cadence User Interface SKILL Functions Reference

Forms

hiSetFormPosition

```
hiSetFormPosition(  
    l_location  
)  
=> t / nil
```

Description

Sets the default initial location for forms displayed without an explicit location argument.

It will always be overridden by specifying a location to [hiDisplayForm](#).

Note: If the user has set the `formPlacement` resource to something other than the default, this location is not used.

The default value can be set in your `.Xdefaults` file with the line

```
Opus.formPlacement: default
```

or by setting the `formPlacement` property of the Command Interpreter Window to "default" with the line

```
hiGetCIWindow()=>formPlacement="default"
```

Arguments

<i>l_location</i>	Location being specified. This must be a list of two integers.
-------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the form position is set.
----------	---

<i>nil</i>	Returns <i>nil</i> and an error message if an error occurs.
------------	---

Cadence User Interface SKILL Functions Reference

Forms

hiSetFormSize

```
hiSetFormSize(  
    r_form  
    l_size  
)  
=> t / nil
```

Description

Resizes a form programmatically.

Arguments

<i>r_form</i>	Handle to the form, which is returned by <u>hiCreateAppForm</u> .
<i>l_size</i>	New size of the form, specified as <i>width:height</i> or a list of width and height.

Value Returned

t	The form was resized.
nil	The form could not be resized.

Cadence User Interface SKILL Functions Reference

Forms

hiSetFormToDefaults

```
hiSetFormToDefaults(  
    r_form  
)  
=> t / nil
```

Description

Sets each field in the form to its default value.

Callbacks for the fields will be called if this results in the value of a field changing.

Arguments

<i>r_form</i>	Form to reset.
---------------	----------------

Value Returned

<i>t</i>	Returns <i>t</i> if the default values are set.
----------	---

<i>nil</i>	Returns <i>nil</i> and an error message if an error occurs.
------------	---

Cadence User Interface SKILL Functions Reference

Forms

hiSetInsertionPosition

```
hiSetInsertionPosition(  
    g_form  
    s_field  
    x_newPosition  
)  
=> t / nil
```

Description

Resets the position of the insertion cursor in a type-in field. You can use the `hiGetInsertionPosition` function to get the current position of the cursor.

Arguments

<i>g_form</i>	Handle to the form, which is returned by hiCreateAppForm .
<i>s_field</i>	Symbol name of the field.
<i>x_newPosition</i>	The new position of the insertion cursor, specified as an integer. The beginning of the field is 0, the first character in the field is 1, and so on. If you specify a number higher than the last character in the field, the cursor is placed at the last character.

Value Returned

t	The position of the insertion cursor was changed.
nil	The position of the insertion cursor could not be changed.

Reference

[hiGetInsertionPosition](#)

Cadence User Interface SKILL Functions Reference

Forms

hiSetLayerCyclicValue

```
hiSetLayerCyclicValue(  
    d_techFileId  
    l_lpp  
    r_form  
)  
=> t / nil
```

Description

Sets the current value of the layer cyclic field.

This function should only be called if the form (*r_form*) contains a layer cyclic field (see [hiCreateLayerCyclicField](#)).

Arguments

<i>d_techFileId</i>	The techfile object associated with the layers of the cyclic field.
<i>l_lpp</i>	List containing the layer name (or number) and purpose. This layer must exist in <i>d_techFileId</i> . (This argument is what is specified as the second argument to dbGetLayer .)
<i>r_form</i>	Form containing the layer cyclic field. This form handle is returned from hiCreateAppForm .

Value Returned

<i>t</i>	Returns <i>t</i> if the value is set.
<i>nil</i>	Returns <i>nil</i> and an error message if an error occurs, or if the layer-purpose pair is not found in the techfile.

hiSetScrollBarValue

```
hiSetScrollBarValue(  
    g_formOrScrollRegion  
    l_newValues  
)  
=> l_values
```

Description

Sets the horizontal and vertical scroll bars in a form or scroll region field to the position you specify, therefore allowing you to display a specific part of the form or scroll region field programmatically. Use `hiGetScrollBarInfo` to get information about the form or scroll region field's scroll bars before you use this function.

Arguments

g_formOrScrollRegion

Handle to the form or scroll region field.

l_newValues

The new values, in pixels, for the scroll bars. Use the following syntax for *l_newValues*:

```
list( [nil | x_horzValue] [nil | x_vertValue]  
)
```

The first element of the list is the new pixel value for the horizontal scroll bar or `nil` to leave the horizontal scroll bar position unchanged. The second element of the list is the new pixel value for the vertical scroll bar or `nil` to leave the scroll bar position unchanged.

To get the minimum and maximum limits for *x_horzValue* and *x_vertValue*, use the [hiGetScrollBarInfo](#) function. The minimum value is usually 0. The maximum value is the size of the form or scroll region field minus the slider size (the visible part of the form or scroll region field).

If you provide a value that is less than the minimum, the minimum value is used. If you provide a value that is more than the maximum (the size of the form or scroll region field minus the slider size), the maximum value is used.

Cadence User Interface SKILL Functions Reference

Forms

If the vertical or horizontal scroll bar is not visible, the new value for that scroll bar is not set.

Value Returned

l_values

The scroll bar values in the following form:

```
list( [nil | x_newHorzValue] [nil |  
x_newVertValue] )
```

The first element of the list is the new position of the horizontal scroll bar or `nil` if the scroll bar is not visible. The second element of the list is the new position of the vertical scroll bar or `nil` if the scroll bar is not visible.

Reference

[hiGetScrollBarInfo](#)

Cadence User Interface SKILL Functions Reference

Forms

hiSetTopListItem

```
hiSetTopListItem(  
    r_listBoxField  
    x_itemPosition  
)  
=> t / nil
```

Description

Sets the first visible item in a list box field to the specified position (*x_itemPosition*). A position value of 1 indicates the first item, a position value of 2 indicates the second item, and so on. A value of 0 (zero) specifies the last item in the list.

This function is only used with the list box fields in forms.

Arguments

<i>r_listBoxField</i>	List box field to reset.
<i>x_itemPosition</i>	The position of the item that is made the first visible item.

Value Returned

t	Returns t on successful completion.
nil	Returns nil if the position is not within the specified range.

Cadence User Interface SKILL Functions Reference

Forms

hiSetListItemVisible

```
hiSetListItemVisible(  
    o_listBoxField  
    x_position  
)  
=> t/nil
```

Description

Brings the specified list box item into view in the list box. Takes a list box field of a form as the first argument and the numeric position of the item you want to bring into view as the second argument. If the numeric position specified is 0 or less than 0, the last item in the list is scrolled into view.

Arguments

<i>o_listBoxField</i>	The list box field.
<i>x_position</i>	Numeric position of the list box item you want to bring into view. The first item is represented by 1. If <i>x_position</i> is 0 or less than 0, the last item in the list is scrolled into view.

Value Returned

<i>t</i>	Always returns <i>t</i> , except when <i>x_position</i> is greater than the number of items in the list box.
<i>nil</i>	Returns <i>nil</i> if <i>x_position</i> is greater than the number of items in the list box.

hiSetListItemCenter

```
hiSetListItemCenter(  
    o_listBoxField  
    x_position  
)  
=> t/nil
```

Description

Brings the specified list box item into view in the center of the list box, if possible. Takes a list box field of a form as the first argument and the numeric position of the item you want to bring into view as the second argument.

If the numeric position specified is 0 or less than 0, the last item in the list is scrolled into view.

Note: If you specify the first or last item in the list, white space is never created above or below the item to bring it into the center of the list box.

Arguments

<i>o_listBoxField</i>	The list box field.
<i>x_position</i>	Numeric position of the list box item you want to bring into view. The first item is represented by 1. If <i>x_position</i> is 0 or less than 0, the last item in the list is scrolled into view.

Value Returned

<i>t</i>	Always returns <i>t</i> , except when <i>x_position</i> is greater than the number of items in the list box.
<i>nil</i>	Returns <i>nil</i> if <i>x_position</i> is greater than the number of items in the list box.

hiShowFieldBorders

```
hiShowFieldBorders(  
    g_onOrOff  
)  
=> t / nil
```

Description

Displays or hides borders around form fields. Used as a debugging aid to avoid overlapping fields when laying out forms.

The default value of the flag is `nil`. The argument `g_onOrOff` can be any non-`nil` value or `nil`. If the argument is non-`nil`, the function sets a global flag that causes borders to be drawn around fields in a form when that form is instantiated. Once a form has been instantiated, the borders remain drawn or undrawn until the form is deleted and re-created.

The color used for drawing the borders is controlled by the X resource `Opus.borderColor`.

Arguments

<i>g_onOrOff</i>	State of the field borders. Valid values: Any non- <code>nil</code> value: borders on <code>nil</code> : borders off
------------------	--

Value Returned

<code>t</code>	Returns <code>t</code> if the value passed in is non- <code>nil</code> .
<code>nil</code>	Returns <code>nil</code> if the value passed in is <code>nil</code> .

hiStoreFormLocation

```
hiStoreFormLocation(  
    s_formName  
    l_location  
)  
=> t
```

Description

Stores a form's location coordinates so that the form can be restored later to the same position.

This function stores the name and associated xy position of a form in an internal table. The function [hiDisplayForm](#) accesses this information prior to displaying the form. The location is updated when the form is moved so that the form does not go back to its old location when closed and reshown. The information stored overrides any positional information specified in the [hiDisplayForm](#) function.

This function is primarily for restoring a saved session.

Arguments

<i>s_formName</i>	Name or base name of the form whose location is to be stored. This argument can contain either the form name or the base name passed by the argument in the hiCreateAppForm function.
<i>l_location</i>	Coordinates to be stored for the specified form.

Value Returned

t	The location of the form is stored successfully.
---	--

Cadence User Interface SKILL Functions Reference

Forms

hiUpdateFormBlock

```
hiUpdateFormBlock(  
    r_form  
)  
=> t / nil
```

Description

Used to reset a form if you have blocked multiple updates with the `hiSetFormBlock` function. Calling `hiUpdateFormBlock(r_form)` is equivalent to calling `hiSetFormBlock(r_form nil)`.

Note: Use the [`hiSetFormBlock`](#) and `hiUpdateFormBlock` functions only if you are making extensive changes to a form.

Arguments

<i>r_form</i>	Handle to the form you want to reset.
---------------	---------------------------------------

Value Returned

<i>t</i>	Returns <i>t</i> unless <i>r_form</i> is not valid.
----------	---

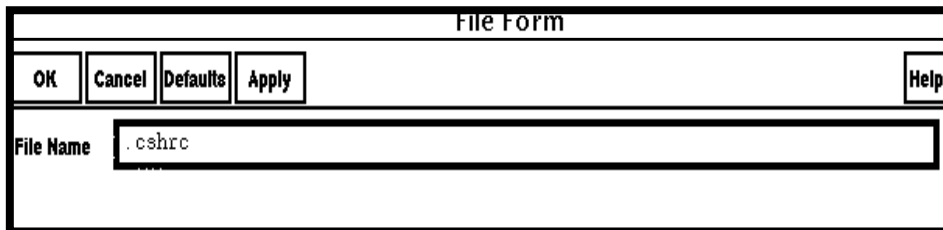
<i>nil</i>	Returns <i>nil</i> if <i>r_form</i> is not valid.
------------	---

Programming Samples

This section includes several programming samples for creating forms.

File Form: Creating String Fields

This sample creates a form with a single string field. The user types the name of a file in the File Name field. The callback checks whether the file exists. If the file exists, the form callback displays it in a Show File window.



```
;;; creating the File Name field
trFileNameField = hiCreateStringField(
    ?name      'trFileNameField
    ?prompt    "File Name"
    ?defValue  ".cshrc"
    ?callback  "trDoFieldCheckCB( hiGetCurrentForm() )"
    ?editable  t
)

;;; checking if the file exists
procedure( trDoFieldCheckCB( theForm )
    if( isFile( theForm->trFileNameField->value )
        then
            println("File exists")
            t
        else
            println("File Does Not Exist--Try Again")
            nil
        ) ;if
    ) ; procedure

;;; creating the form
trFileForm = hiCreateAppForm(
    ?name      'trFileForm
    ?formTitle "File Form"
    ?callback  'trFileFormCB
    ?fields    list( trFileNameField )
    ?unmapAfterCB t
)

procedure( trFileFormCB( theForm )
    if( trDoFieldCheckCB( theForm )
```

Cadence User Interface SKILL Functions Reference

Forms

```
    then
        hiSetCallbackStatus( theForm t )
        hiHighlightField( theForm 'trFileNameField 'background )
        view( theForm->trFileNameField->value )
    else
        hiSetCallbackStatus( theForm nil )
        hiHighlightField( theForm 'trFileNameField 'highlight )
    ) ; if
) ; procedure

;;; displaying the form
hiDisplayForm( trFileForm )
```


Form Fields: Creating Fields in a Form

The following sample displays different types of fields in a form.

Sample Form

OK Cancel Defaults Apply Help

Cycle Through: 3

Boolean ☒

Button Box Do a Do b Do c

Cone Size: ☒ small ☐ medium ☐ large

Slide 500

Label

```
;;; creating the list of items in the cyclic field
trCyclicList = '(
  "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"
  "11" "12" "13" "14" "15" "16" "17" "18" "19" "20" "21" "22"
)

;;; creating the cyclic field
trCyclicField = hiCreateCyclicField(
  ?name      'trCyclicField
  ?prompt    "Cycle Through: "
  ?choices   trCyclicList
  ?value     "3"
  ?defValue  "7"
  ?callback  "println"
)

;;; creating the boolean button field.
;;; The callback for the trBooleanButton field dynamically retrieves the new value
;;; of the field and embeds it in a message
trBooleanButton = hiCreateBooleanButton(
  ?name      'trBooleanButton
  ?buttonText "Boolean"
  ?value     t
  ?defValue  nil
  ?callback  "println( hiGetCurrentForm()->trBooleanButton-> value )"
)

;;; creating the button box field
trButtonBoxField = hiCreateButtonBoxField(
  ?name      'trButtonBoxField
  ?prompt    "Button Box"
  ?choices   ("Do a" "Do b" "Do c")
  ?callback  ("println( 'a )" "println( 'b )" "println( 'c)" )
)

;;; creating the radio field.
;;; The callback for the trConeRadioField field dynamically retrieves the new value
```

Cadence User Interface SKILL Functions Reference

Forms

```
;;; of the field and imbeds it in a message
trConeRadioField = hiCreateRadioField(
    ?name      'trConeRadioField
    ?prompt    "Cone Size: "
    ?choices   list( "small" "medium" "large" )
    ?value     "small"
    ?defValue  "large"
    ?callback  '( "printf(  \n\n%s cone chosen \n" hiGetCurrentForm()
->trConeRadioField->value )" )
)

;;; creating the scale field
trScaleField = hiCreateScaleField(
    ?name      'trScaleField
    ?prompt    "Slide "
    ?value     500
    ?defValue  250
    ?callback  "println(\"scale changed \")"
    ?range     0:750
)

;;; creating the label field
trLabelField = hiCreateLabel(
    ?name      'trLabelField
    ?labelText "Label"
    ?justification CDS_JUSTIFY_RIGHT
)

;;; creating the form
hiCreateAppForm(
    ?name      'trSampleForm
    ?formTitle "Sample Form"
    ?callback  "println( 'FormAction )"
    ?fields
        list(
            trCyclicField
            trBooleanButton
            trButtonBoxField
            trConeRadioField
            trScaleField
            trLabelField
        )
    ?unmapAfterCB t
) ; hiCreateAppForm

;;; displaying the form
hiDisplayForm( trSampleForm )
```

Window Form: Creating an Application

The following sample creates a form that allows the user to change the title or size of a given window.

The image shows a standard Mac OS-style dialog box titled 'window:1'. It contains five buttons at the top: 'OK', 'Cancel', 'Defaults', 'Apply', and 'Help'. Below the buttons are three input fields. The first field is labeled 'Name' and contains the text 'Window Form Sample'. The second field is labeled 'Height' and contains the number '304'. The third field is labeled 'Width' and contains the number '753'.

```

;;; retrieving the height of the bounding box
procedure( trGetBBoxHeight( bBox )
  let( ( ll ur lly ury )
    ll = lowerLeft( bBox )
    lly = yCoord( ll )
    ur = upperRight( bBox )
    ury = yCoord( ur )
    ury - lly
  ) ; let
) ; procedure

;;; retrieving the width of the bounding box
procedure( trGetBBoxWidth( bBox )
  let( ( ll ur llx urx )
    ll = lowerLeft( bBox )
    llx = xCoord( ll )
    ur = upperRight( bBox )
    urx = xCoord( ur )
    urx - llx
  ) ; let
) ; procedure

;;; retrieving the window height
procedure( trGetWindowHeight( wid )
  trGetBBoxHeight( hiGetAbsWindowScreenBBox( wid ) )
) ; procedure

;;; retrieving the window width
procedure( trGetWindowWidth( wid )
  trGetBBoxWidth( hiGetAbsWindowScreenBBox( wid ) )
) ; procedure

;;; resizing the window to the given bounding box
procedure( trResizeWindow( @key ( id hiGetCurrentWindow() ) ( height nil ) ( width
nil ) )
  let( ( bBox ll ur llx lly urx ury urNew )
    bBox = hiGetAbsWindowScreenBBox( id )
    ll = lowerLeft( bBox )
    ur = upperRight( bBox )

```

Cadence User Interface SKILL Functions Reference

Forms

```
llx = xCoord( ll )
lly = yCoord( ll )
urx = xCoord( ur )
ury = yCoord( ur )

unless( height height = ury - lly )
unless( width width = urx - llx )

urNew = llx + width : lly + height

hiResizeWindow( id list( ll urNew ) )

id
) ; let
) ; procedure

;;; creating a form that acts upon the window with the
;;; window id (wid)
procedure( trCreateWindowForm( wid )
  let( ( nameField heightField widthField theFormSymbol theForm )

    nameField =
      hiCreateStringField(
        ?prompt "Name"
        ?name 'nameField
        ?value hiGetWindowName( wid )
        ?defValue hiGetWindowName( wid )
        ?callback "println( 'nameField )"
      )

    heightField =
      hiCreateScaleField(
        ?prompt "Height"
        ?name 'heightField
        ?value trGetWindowHeight( wid )
        ?defValue trGetWindowHeight( wid )
        ?range list( 0 yCoord( hiGetMaxScreenCoords() ) )
        ?callback "println( 'heightField )"
      )

    widthField =
      hiCreateScaleField(
        ?prompt "Width"
        ?name 'widthField
        ?value trGetWindowWidth( wid )
        ?defValue trGetWindowWidth( wid )
        ?range list( 0 xCoord( hiGetMaxScreenCoords() ) )
        ?callback "println( 'widthField )"
      )

    ;; specifying a unique symbol to allow multiple
    ;; instances of the form to be active at a single
    ;; time
    theFormSymbol = gensym( 'WindowForm )

    ;; building the form's data structure
    theForm =
      hiCreateAppForm(
        ?name theFormSymbol
```

Cadence User Interface SKILL Functions Reference

Forms

```
?formTitle sprintf( nil "%L" wid )
?callback "trWindowFormCB( hiGetCurrentForm())"
?fields list( nameField heightField widthField)
?unmapAfterCB t
)

;;; associating the wid with the form via a user
;;; defined slot
theForm->wid = wid

theForm  ;;; the return value

) ; let
) ; procedure

;;; The following callback for the form handles the
;;; renaming and resizing of the window.
procedure( trWindowFormCB( theForm )
  let( ( wid newName newHeight newWidth )
    ;;; picking up the current form values
    wid = theForm->wid
    ;;; user specified slot linking the form to the window
    newName = theForm->nameField->value
    ;;; accessing the current heightField
    newHeight = theForm->heightField->value
    ;;; accessing the current widthField
    newWidth = theForm->widthField->value

    ;;; setting the window name
    hiSetWindowName( wid newName )

    ;;; resizing the window
    trResizeWindow(
      ?id wid
      ?height newHeight
      ?width newWidth
    ) ; trResizeWindow

    t      ;;; the return value

  ) ; let
) ; procedure
```

Testing Your Application

To test your application using the CIW, type:

```
wf1 = trCreateWindowForm( window(1))
hiDisplayForm(wf1)
```

After the form is displayed, change the values of the Height and Width fields and type a new title for the CIW in the Name field. Select the Apply button to see the changes in the title and size of the CIW.

List Editor: Creating List Box Fields

The following sample shows you how to include a list box in a form. The user can rearrange the items of the list box by selecting the Move Up or Move Down buttons.

List Editor	
OK	Cancel
Defaults	Apply
Help	
Items	<div style="border: 1px solid black; min-height: 100px; margin-bottom: 5px;"> 1 2 3 </div> <div style="text-align: center; margin-bottom: 5px;">Move Up</div> <div style="text-align: center; margin-bottom: 5px;">Move Down</div> <div style="text-align: center;">Delete</div>

```

;;; The following procedures returns a copy of aList
;;; with elem swapped one position closer to the top
;;; of the list. == aList if elem is already on top
;;; of the list.
procedure( trSwapElementTowardTop( elem aList )
  cond(
    ( !aList nil )
    ( elem == car( aList ) aList )    ;;; already the first item
    ( elem == cadr( aList )
      `( ,elem ,car( aList ) ,@cddr( aList ) )
    )
    ( t
      cons(
        car( aList )
        trSwapElementTowardTop( elem cdr( aList ) )
      )
    ) ; t
  ) ; cond
) ; procedure

```

```

/*
trSwapElementTowardTop( 1 nil )
trSwapElementTowardTop( 1 '( 1 ) )
trSwapElementTowardTop( 1 '( 1 2 ) )
trSwapElementTowardTop( 1 '( 2 1 ) )
trSwapElementTowardTop( 1 '( 2 1 3 4 ) )
trSwapElementTowardTop( 1 '( 2 3 4 1 ) )
trSwapElementTowardTop( 1 '( 2 1 3 4 ) )
*/

```

```

;;; The following procedure returns a copy of aList with
;;; elem swapped one position closer to the bottom of
;;; the list.==aList if elem is already on the bottom of
;;; the list.

```

```

procedure( trSwapElementTowardBottom( elem aList )
  cond(

```

Cadence User Interface SKILL Functions Reference

Forms

```
( !aList nil )
( length( aList ) == 1 aList )
( elem == car( aList ) ;
  `( ,cadr( aList ) ,elem ,@cddr( aList ))
)
( t
  cons(
    car( aList )
    trSwapElementTowardBottom( elem cdr( aList ))
  )
) ; t
) ; cond
) ; procedure

/*
trSwapElementTowardBottom( 1 nil )
trSwapElementTowardBottom( 1 '( 1 ) )
trSwapElementTowardBottom( 1 '( 1 2 ) )
trSwapElementTowardBottom( 1 '( 2 1 ) )
trSwapElementTowardBottom( 1 '( 2 1 3 4 ) )
trSwapElementTowardBottom( 1 '( 2 3 4 1 ) )
*/

;;; The following procedure returns a list build using the
;;; elements of aList as indices into anAssocList. It looks
;;; up each element of aList in anAssocList to determine the
;;; element to include in the result.
procedure( trReorderList( anAssocList aList )
  foreach( mapcan item aList
    list( cadr( assoc( item anAssocList )) )
  ) ; foreach
) ; procedure

;;; A list box field can only display text strings.
;;; Since trListEditor should work on any list, the
;;; code should translate back and forth from a data
;;; item X to its print representation returned by
;;; sprintf( nil "%L"X).
procedure( trListEditor_Choices( aList )
  foreach( mapcar item aList
    sprintf( nil "%L" item )
  ) ; foreach
) ; procedure

;;; The following procedure recovers the underlying items
;;; from their print representations.
procedure( trListEditor_AssocList( aList )
  foreach( mapcar item aList
    list( sprintf( nil "%L" item ) item )
  ) ; foreach
) ; foreach

procedure( trListEditor( aList )
  let( ( listBoxField buttonBoxField aListChoices theForm moveUpButton
    moveDownButton )

    listBoxField = hiCreateListBoxField(
      ?name 'trListBoxField
      ?prompt "Items"
      ?choices trListEditor_Choices( aList )
    )
  )
)
```

Cadence User Interface SKILL Functions Reference

Forms

```
?numRows min( length( aList ) 10 )
)

;;; creating the Move Up button
moveUpButton = hiCreateButton(
  ?name 'trMoveUpButton
  ?buttonText "Move Up"
  ?callback "trListEditorCB( hiGetCurrentForm()
'MoveUp )"
)

;;; creating the Move Down button
moveDownButton = hiCreateButton(
  ?name 'trMoveDownButton
  ?buttonText "Move Down"
  ?callback "trListEditorCB( hiGetCurrentForm() 'MoveDown)"
)

;;; creating the Delete button
deleteButton = hiCreateButton(
  ?name 'trDeleteButton
  ?buttonText "Delete"
  ?callback "trListEditorCB( hiGetCurrentForm() 'Delete )"
)

;;; creating the form
theForm = hiCreateAppForm(
  ?name gensym( 'trListEditor )
  ?formTitle "List Editor"
  ?fields list( listBoxField moveUpButton moveDownButton deleteButton )
  ?callback "trListEditorCB( hiGetCurrentForm()
'Apply )"
)

theForm->trListEditor_AssocList = trListEditor_AssocList( aList )
;;; displaying the form
hiDisplayForm( theForm )
) ; let
) ; procedure

procedure( trListEditorCB( theForm theAction )
  prog( ( selectedItem choices )
    selectedItem = car( theForm->trListBoxField->value )
    when( !selectedItem
      printf( "You haven't selected an item yet\n" )
      return( nil )
    ) ; when
    choices = theForm->trListBoxField->choices
    case( theAction
      ( MoveUp
        theForm->trListBoxField->choices =
trSwapElementTowardTop(
          selectedItem      ;;; selected element
          choices           ;;; within list of items
        )
        theForm->trListBoxField->value = list(
selectedItem )
      ) ; MoveUp
      ( MoveDown
        theForm->trListBoxField->choices =
trSwapElementTowardBottom(
```


Cadence User Interface SKILL Functions Reference

Forms

```
selectedItem      ;;; selected element
choices           ;;; within list of items
)
theForm->trListBoxField->value = list(
selectedItem )
) ; MoveDown
( Delete
  theForm->trListBoxField->choices = remove(
    selectedItem      ;;; selected element
    choices           ;;; from list of items
  ) ; remove
  ;;; no selection
) ; Delete
( Apply
  println( theForm->trListEditor_result =
trListEditor_ReorderList( theForm ))
  ) ; Apply
( t
  error( "Unsupported action: %L" theAction )
  ) ; t
) ; case
) ; let
) ; procedure

procedure( trListEditor_ReorderList( theForm )
let( ( assocList choices )
  assocList = theForm->trListEditor_AssocList
  choices = theForm->trListBoxField->choices
  trReorderList( assocList choices )
  ) ; let
) ; procedure
```

Testing Your Application

To test your list editor, type:

```
Z = trListEditor( '( 1 2 3 ) )
```

After the form is displayed, rearrange the list box by moving the items up or down.

File Browser: Creating Multiline Text Fields

The following sample uses a multiline text field to display the contents of a given file.

```
procedure( trMoreFile( )
  let( ( fileNameField fileContentsField theForm )
  ;; creating the File Name string field
    fileNameField = hiCreateStringField(
      ?name 'trFileNameField
      ?prompt "File Name"
      ?defValue ".cshrc"
      ?callback "trMoreFileCB( hiGetCurrentForm() 'trFileNameField )"
    )

  ;; creating the Contents multiline text field for
  ;; viewing the file
    fileContentsField = hiCreateMLTextField(
      ?name 'trMLTextField
      ?prompt "Contents"
      ?defValue ""
      ?editable t ;; so users can edit the file
    )

  ;; creating the form
    theForm = hiCreateAppForm(
      ?name gensym( 'trMoreFile )
      ?formTitle "File Browser"
      ?callback "trMoreFileCB(hiGetCurrentForm() 'apply)"
      ?buttonLayout 'ApplyCancelDef
      ?fields
        list( fileNameField fileContentsField )
    )
    hiDisplayForm( theForm )
  ) ; let
) ; procedure

procedure( trMoreFileCB( theForm actionOrField )
  case( actionOrField
    ;; checking if the file exists
    ( trFileNameField
```

Cadence User Interface SKILL Functions Reference

Forms

```
if( isFile( theForm->trFileNameField->value )
    then
        println("File exists")
        t
    else
        println("File Does Not Exist--Try Again")
        nil
    ) ;if
)
( apply
  when( trMoreFileCB( theForm 'trFileNameField )
    theForm->trMLTextField->value =
      trReadFileIntoString(
theForm->trFileNameField->value )
    ) ; when
  )
  ( t
    error( "Illegal action or unknown field: %L" actionOrField )
  )
) ; case
) ; procedure

;;; The following procedure returns a list of the lines
;;; in the file in the correct order
procedure( trReadFile( pathname )
  let( ( inPort nextLine allLines )
    inPort = infile( pathname )
    when( inPort = infile( pathname )
      while( gets( nextLine inPort )
        allLines = cons( nextLine allLines )
      ); while
      close( inPort )
    ) ; when
    reverse( allLines )
  ) ; let
) ; procedure

;;; reading the entire file into a single string
procedure( trReadFileIntoString( pathname )
  let( ( allLines )
    allLines = trReadFile( pathname )
    when( allLines apply( 'strcat allLines ) )
  ) ; let
) ; procedure
```

Testing Your Application

To test your file browser, type:

```
trMoreFile
```

After the file browser is displayed, type the name of an existing file and press Apply to view the file.

Cadence User Interface SKILL Functions Reference
Forms

Windows

The following topics are discussed in this chapter:

- [Window Management](#) on page 422
- [Window Functions](#) on page 424
- [Viewing Functions](#) on page 489
- [Splash Screen Functions](#) on page 509

Window Management

A window is a rectangular area on the screen that lets you view graphical output. Client applications can display overlapping, nonoverlapping, and nested windows. A window can be displayed on one or more screens and on one or more hardware platforms.

Note: The functions specified in this chapter are not available in Concept SKILL.

All windows have

- Border width of zero or more pixels
- Optional background
- Input mask
- Event suppression mask
- Property list

You can specify the window border and banner colors.

All windows except the root have a parent and are clipped by their parent window. If a window is stacked on top of another window, it obscures that other window for the purpose of input and output. Attempts to output to the obscured area of a window are not successful, and input events (for example, pointer motion) are not generated for the obscured area.

Output to a window is always clipped to the inside of the window. Therefore, graphics operations never affect the window border. Borders are added to the window size specified.

An external (separate process) window manager can override your choice as to size, border width, and position for a window. Your program must use the actual size and position of the top window that is reported when the window is first mapped. The window manager can also be used for window movement, resizing, iconifying, and so forth.

Each application can create its own window to be used to identify the application and to serve as a work space for graphics output and user interaction. When a window is created and displayed, a unique window number is generated and displayed in the right corner of the banner. A menu bar on the left side of the window banner can contain user-defined pulldown menus. Each window also contains a *Help* button, displayed to the left of the window number.

Although there might be several application windows open on the screen at the same time, there is only one current window—the window where the last command was executed. Depending on how the window focus mechanism is set, the current window can be set by moving the cursor into a window or by clicking the mouse button in a window (see [hiFocusToCursor](#)).

Cadence User Interface SKILL Functions Reference

Windows

The active window is indicated by a colored window number or a colored label banner if it exists. All inactive windows have background colored banners. More than one window can display graphics at one time; however, only the current window receives user interaction. The Command Interpreter Window can never be set as the current window.

Window Functions

This section describes the functions available for manipulating windows.

getCurrentWindow

The `getCurrentWindow` function has been replaced by the [hiGetCurrentWindow](#) function.

getMaxScreenCoords

The `getMaxScreenCoords` function has been replaced by the [hiGetMaxScreenCoords](#) function.

hiGetDbuPoint

```
hiGetDbuPoint(  
    w_window  
)  
=> l_point / nil
```

Description

Returns the current location of the pointer, or `nil` if the pointer is not within the drawable area of the window.

If the pointer is currently over another window that obscures the window in question, it is considered within the bounds of the window if it would be in the window if the window were on top.

Before returning the current location of the pointer, `hiGetDbuPoint` calls the appropriate transform and adjust procedures.

Arguments

<code>w_window</code>	Window ID.
-----------------------	------------

Value Returned

<code>l_point</code>	Location of the pointer in database units.
<code>nil</code>	Returns <code>nil</code> if the pointer is not within the drawable area of the window.

Reference

[`hiGetDrawThruDelta`](#), [`hiGetDbuPoint`](#), [`hiGetScreenPoint`](#)

hiGetDrawThruDelta

```
hiGetDrawThruDelta( )  
=> x_delta
```

Description

Returns the minimum number of pixels that the pointer must move before a drag (also known as DrawThru) is recognized as having been started if a mouse button is held down.

This value is also the maximum number of pixels that the pointer can be moved between two successive clicks for those clicks to be recognized as a double click.

Arguments

None.

Value Returned

x_delta Minimum number of pixels the pointer must move.

Reference

[hiGetDbuPoint](#), [hiGetScreenPoint](#)

hiGetScreenPoint

```
hiGetScreenPoint(  
    w_window  
)  
=> l_point / nil
```

Description

Returns the current location of the pointer, or `nil` if the pointer is not within the drawable area of the window.

Location is given in pixels relative to the upper left corner of the drawable area.

If the pointer is currently over another window that obscures the window in question, it is considered within the bounds of the window if it would be in the window if the window were on top.

Arguments

<code>w_window</code>	ID of the window.
-----------------------	-------------------

Value Returned

<code>l_point</code>	Location of the pointer in pixels relative to the upper left corner of the drawable area.
<code>nil</code>	Returns <code>nil</code> if the pointer is not within the drawable area of the window.

Reference

[hiGetDrawThruDelta](#), [hiGetDbuPoint](#)

hiOpenWindow

```
hiOpenWindow(  
    [?bBox l_boxSpec]  
    [?type t_widgetType]  
    [?appType t_applicationType]  
    [?menus l_menus]  
    [?labels l_labels]  
    [?help t_helpString]  
    [?scroll g_scrollbars]  
    [?form g_form]  
    [?closeProc s_closeProc]  
    [?iconPosition l_iconPosition]  
)  
=> w_windowId / nil
```

Description

Creates and displays a window.

Note: This function is not available in Concept SKILL.

After this window has been created, it becomes the current window.

If you do not want this window ever to become the current window, you can set the window property `neverCurrentWindow` to `t`:

```
window->neverCurrentWindow = t
```

When this property is set on a current window, the current window will be reset to the **last** current window or to another open window. If another window cannot be found, the current window will be reset to `nil`.



Caution

Setting this property can affect previously saved replay files that contain this window, and these log files might not replay correctly.

Arguments

<code>l_boxSpec</code>	Lower left and upper right corners of the window in screen coordinates (assuming 0:0 to be the lower left corner of the screen). The 'default' and 'interactive' symbols are also acceptable values for the <code>l_boxSpec</code> argument. An icon position can also be specified (<code>l_iconPosition</code>).
------------------------	--

Cadence User Interface SKILL Functions Reference

Windows

If *l_boxSpec* is *nil*, either you are prompted with a rubberbanding box to enter a window size, or a default size is used, depending upon the value of the global variable *hivWinStyle*.

t_widgetType One of the following strings: *graphics*, *text*, *encap*, *form*, *browser*, or *none* (plot windows) representing the type of window widget to be created. If *widgetType* is not specified, the default is *graphics*.

t_applicationType String representing the window application type or name. This string is also used as the window's bindkey prefix. If not specified, *t_applicationType* defaults to *" "*.

l_menus List of menus to appear in the window banner. Each menu should be either the value returned from a call to [*hiCreatePulldownMenu*](#) or the menu symbol representing that value. These menus are placed from left to right on the window banner in the order listed. A maximum of 20 menus can be placed on each window. Each menu can be placed only once on a window banner.

l_labels List of strings to appear in the label banner. These labels are placed from left to right on the label banner in the order listed. A maximum of 4 labels can be placed on each window.

t_helpString String used as one of the keywords that Help logic uses when it is constructing the names of the various help files and/or help symbols searched for in the help files. If you do not specify this argument or specify *" "* (an empty string), the help argument is set to the *appID* (or *appName*).

The help string for a window can also be set at any time using the *hiSetHelp* or *hiSetWindowsAtts* functions, and can be retrieved with the *hiGetHelp* function.

For Cadence application windows, the application may modify the help string at any time.

When help is invoked in a window, the help string is used for the help symbol that is looked up in the *.HRF* file (for HRF help) or the *.tgf* file (for *.tgf* help).

Cadence User Interface SKILL Functions Reference

Windows

The help string can also be used to modify the `appName` used for help for the window by assigning an `hiHelpAppName` property to the symbol representation of the string.

For example:

```
putprop( concat( help_string ) app_name_string 'hiHelpAppName)
```

will set the `appID` used for Help whenever *help_string* is used as the help string for any window to *app_name_string*.

g_scrollbars

Either `t` or `nil` to specify the use of scroll bars. This field is valid only for windows with *widgetType* equal to `graphics`. If set to `t`, scroll bars appear on the right and bottom of the window. If not specified, the property `useScrollbars` from the CIW is used. This value can be set from the User Preferences form.

g_form

SKILL representation of the form returned by `hiCreateAppForm` or `hiCreateForm` and displayed in a window. This field is valid only for windows with *widgetType* equal to `form`. When the window containing the form is closed, the form is NOT destroyed. It can be redisplayed in another window.

S_closeProc

String or symbol for the name of the procedure to be performed when the specified window is closed. This closing procedure is strictly for clean up and must not call `hiCloseWindow`, any enter functions, or any blocking functions that display forms or dialog boxes. Because this clean-up procedure is called no matter how you close the window (from a window manager or through Cadence software), you might need to delete existing calls to clean-up functions from the user interface.

Note: A C close procedure is called before any SKILL close procedures. If the C procedure returns `nil`, none of the SKILL close procedures are called and the window remains open.

l_iconPosition

Icon location (in X Window System screen coordinates) for the window. This causes the window to be iconified initially. If *l_iconPosition* is `nil` or not specified, the window is not iconified initially.

In addition to the `?iconPosition` keyword argument, the *l_boxSpec* argument can also contain the icon position. If both

Cadence User Interface SKILL Functions Reference

Windows

methods are used, `?iconPosition` overrides the position passed in `l_boxSpec` unless `l_iconPosition` is `nil`.

Value Returned

<code>w_windowId</code>	Returns the window ID of the created window.
<code>nil</code>	Returns <code>nil</code> and issues an error message if the window is not created.

Example

```
hiOpenWindow(?bBox list(0:0 300:300)
?type "graphics"
?menus list(hiOpenMenu 'hiDMMenu)
?labels '("label 1" "label 2")
?scroll t
)
=> window:2

hiOpenWindow(?bBox 'interactive
?type "form"
?appType "formdow"
?form hiCreateForm('myform "My Form" ""
list(hiCreateStringField(?name 's ?prompt "Enter name:"))
"" )
)
=> window:3
```

hiCreateWindow

```
hiCreateWindow(  
    g_boxSpec  
    t_widgetType  
    t_applicationType  
    [t_help]  
    [g_scrollbars]  
    [g_form]  
    [l_iconPosition]  
    )  
=> w_windowId / nil
```

Description

Creates but does not display a window.

Note: This function is not available in Concept SKILL.

After this window has been successfully created, it becomes the current window. If you do not want this window ever to become the current window, you can set the window property `neverCurrentWindow` to `t`:

```
(window->neverCurrentWindow = t)
```

When this property is set on a current window, the current window is reset to the **last** current window or to another open window. If another window cannot be found, the current window is reset to `nil`.



Caution

Setting this property might affect previously saved replay files that contain this window, and these log files might not replay correctly.

Arguments

g_boxSpec

Lower left and upper right corners of the window in screen coordinates (assuming 0:0 to be the lower left corner of the screen). The 'default and 'interactive symbols are also acceptable values for the *g_boxSpec* argument. An icon position can also be specified. (See *l_iconPosition*.) If *g_boxSpec* is `nil`, either you are prompted with a rubberbanding box to enter a window size or a default size is used, depending upon the value of the global variable `hivWinStyle`.

Cadence User Interface SKILL Functions Reference

Windows

<i>t_widgetType</i>	One of the following strings: <code>graphics</code> , <code>text</code> , <code>encap</code> , <code>form</code> , <code>browser</code> , or <code>none</code> (plot windows) that represent the type of window widget to be created.
<i>t_applicationType</i>	<p>String representing the window application type or name. This string is also used as the window's bindkey prefix.</p> <p>Note: For encapsulation windows, if you specify "" (an empty string) for this argument, the string "encap" will be used as the <i>applicationType</i>.</p>
<i>t_help</i>	<p>String used as one of the keywords that Help logic uses when it is constructing the names of the various help files and/or help symbols searched for in the help files. If you do not specify this argument or specify "" (an empty string), the help argument is set to the <code>appID</code> (or <code>appName</code>).</p> <p>The help string for a window can also be set at any time using the <code>hiSetHelp</code> or <code>hiSetWindowsAtts</code> functions, and can be retrieved with the <code>hiGetHelp</code> function.</p> <p>For Cadence application windows, the application may modify the help string at any time.</p> <p>When help is invoked in a window, the help string is used for the help symbol that is looked up in the <code>.HRF</code> file (for HRF help) or the <code>.tgf</code> file (for <code>.tgf</code> help).</p> <p>The help string can also be used to modify the <code>appName</code> used for help for the window by assigning an <code>hiHelpAppName</code> property to the symbol representation of the string.</p> <p>For example,</p> <pre>putprop(concat(help_string) app_name_string 'hiHelpAppName)</pre> <p>will set the <code>appID</code> used for Help whenever <i>help_string</i> is used as the help string for any window to <i>app_name_string</i>.</p>
<i>g_scrollbars</i>	Either <code>t</code> or <code>nil</code> to specify the use of scroll bars. This field is valid only for windows with <code>widgetType</code> equal to <code>graphics</code> . If set to <code>t</code> , scroll bars appear on the right and bottom of the window. If not specified, the property <code>useScrollbars</code> from the CIW is used. This value can be set from the User Preferences form.

Cadence User Interface SKILL Functions Reference

Windows

g_form SKILL representation of the form returned by [hiCreateAppForm](#) and displayed in a form window; valid only for windows with *widgetType* equal to *form*. When the window containing the form is closed, the form is NOT destroyed. The form can be redisplayed in another window.

l_iconPosition Icon location (in X Window System screen coordinates) of the window. This causes the window to be iconified initially. If *l_iconPosition* is *nil* or not specified, the window is not iconified initially.

The *g_boxSpec* argument can also contain the icon position. If both methods are used, this icon position overrides the position passed in *g_boxSpec* unless *l_iconPosition* is *nil*.

Value Returned

w_windowId Returns the window ID of the created window.

nil Returns *nil* and issues an error message if the window is not created.

Example

```
hiCreateWindow('default "graphics" "my app")  
=> window:4
```

hiDisplayWindow

```
hiDisplayWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Displays the window created by [hiCreateWindow](#).

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to display.
-------------------	-----------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the window is displayed.
----------	--

<i>nil</i>	Returns <i>nil</i> and issues an error message if the window is not displayed.
------------	--

Cadence User Interface SKILL Functions Reference

Windows

hiGetHelp

```
hiGetHelp(  
    w_windowId  
)  
=> t_help
```

Description

Retrieves the help string that is defined for the window.

Arguments

<i>w_windowId</i>	The window whose help string you want to retrieve.
-------------------	--

Value Returned

<i>t_help</i>	The help string that is defined for the window.
---------------	---

Cadence User Interface SKILL Functions Reference

Windows

window

```
window(  
    w_windowNumber  
)  
=> w_windowId / nil
```

Description

Retrieves the window identity of a window number.

The window number is the number displayed in the upper-right corner of the window banner. Valid window numbers are positive nonzero integers. The Command Interpreter Window is window number 1. As windows are created (via [hiCreateWindow](#) or [hiOpenWindow](#)), they are given increasingly larger window numbers.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowNumber</i>	Number of the window whose identity you want.
-----------------------	---

Value Returned

<i>w_windowId</i>	Returns the <i>w_windowId</i> associated with the given <i>x_windowNumber</i> .
-------------------	---

<i>nil</i>	Returns <i>nil</i> if there is no window associated with the specified window number.
------------	---

Example

```
hiSetCurrentWindow(window(5))  
=> t
```

Cadence User Interface SKILL Functions Reference

Windows

wtypep

```
wtypep(  
    g_arg  
)  
=> t / nil
```

Description

This predicate function returns *t* if the passed argument is a window type, *nil* otherwise.

Arguments

<i>g_arg</i>	Argument to be checked.
--------------	-------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the argument is a window type.
----------	--

<i>nil</i>	Returns <i>nil</i> if the argument is not a window type.
------------	--

Cadence User Interface SKILL Functions Reference

Windows

windowp

```
windowp(  
    g_object  
)  
=> t / nil
```

Description

Checks whether the specified object is a valid, opened window.

Note: This function is not available in Concept SKILL.

Arguments

<i>g_object</i>	Object you wish to check.
-----------------	---------------------------

Value Returned

t	Returns t if <i>g_object</i> is a valid open window.
---	--

nil	Returns nil if <i>g_object</i> is not a valid open window.
-----	--

hIsWindowSpecifier

```
hIsWindowSpecifier(  
    g_winspec  
)  
=> g_winspec / nil
```

Description

Validates a window specifier, which is passed as the `?bBox` argument to the `hiOpenWindow` function and as the `g_boxSpec` argument to the `hiCreateWindow` function.

Arguments

<i>g_winspec</i>	Window specifier. Must be one of the following values: <ul style="list-style-type: none">■ 'default■ 'interactive■ A bounding box: <code>list(x1:y1 x2:y2)</code> where <i>x1</i> is the lower left x coordinate, <i>y1</i> is the lower left y coordinate, <i>x2</i> is the upper right x coordinate, and <i>y2</i> is the upper right y coordinate.■ A list of bounding box and iconify state: <code>list(bBox iconifyState)</code> where <i>iconifyState</i> is <code>nil</code> or a point.
------------------	--

Value Returned

<i>g_winspec</i>	Returns window specifier if <i>g_winspec</i> is a valid window specifier.
<i>nil</i>	Returns <i>nil</i> if <i>g_winspec</i> is not a valid window specifier.

hiSetWindowAtts

```
hiSetWindowAtts(  
    [?window w_windowId]  
    [?appType t_applicationType]  
    [?help t_help]  
    [?scroll g_scrollbars]  
)  
=> t / nil
```

Description

Changes the application type, help string, or scroll bar attributes of a window. You must have registered the new application type before using this function.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act. If the argument is not specified, the current window is used.
<i>t_applicationType</i>	<p>String representing the window application type or name.</p> <p>You must have registered the new application type before using this function. For example, if you are setting an application type to "chpr", you must have registered it using the command:</p> <pre>hiRegisterBindKeyPrefix("chpr" "encap")</pre> <p>For more information about registering the application type, see <u>hiRegisterBindKeyPrefix</u>.</p>
<i>t_help</i>	String used as one of the key words that Help logic uses when it is constructing the names of the various help files and/or help symbols searched for in the help files. Refer to Online Help documentation for more information. If this argument is not specified, it defaults to the same value as <i>t_applicationType</i> .
<i>g_scrollbars</i>	Either <i>t</i> or <i>nil</i> to specify the use of scroll bars. This field is valid only for <i>graphics</i> widget types. If set to <i>t</i> , scroll bars appear on the right and bottom of the window. If not specified, the property <i>useScrollbars</i> from the CIW is used.

Cadence User Interface SKILL Functions Reference

Windows

Value Returned

<code>t</code>	Returns <code>t</code> if <i>applicationType</i> and/or <i>help</i> are changed. It returns <code>t</code> whether <i>g_scrollbars</i> is set or not.
<code>nil</code>	Returns <code>nil</code> if either <i>applicationType</i> or <i>help</i> is not set. If you are setting both <i>applicationType</i> and <i>help</i> at the same time, the returned value will be <code>nil</code> even if only one of these attributes is not set.

Cadence User Interface SKILL Functions Reference

Windows

hiSetWinStyle

```
hiSetWinStyle(  
    s_style  
)  
=> t / nil
```

Description

Sets the window creation style.

Legal values are 'default, 'interactive and 'bBox.

Arguments

<i>s_style</i>	'default, 'interactive or 'bBox. 'default means new windows will appear with the size and location specified programmatically. 'interactive means you must place the window yourself interactively for new windows. 'bBox is an alternate value for setting 'interactive.
----------------	---

Value Returned

t	Returns t if the style is set.
nil	Returns nil if the operation failed.

Reference

[hiCreateWindow](#), [hiDisplayWindow](#), [hiOpenWindow](#)

hiCloseWindow

```
hiCloseWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Closes and deletes a window.

Note: This function is not available in Concept SKILL.

This function is used for a window created by a call to [hiCreateWindow](#), [hiOpenWindow](#) or [deOpen](#), or for one returned by [hiGetCurrentWindow](#).

If the window is of widgetType `form`, the form contained in this window is NOT destroyed when the window is closed.

Note: See the description of the [hiRegCloseProc](#) and [hiUnregCloseProc](#) functions to register a closing procedure for a window.

If the window closed is the current window, an attempt is made to make an equivalent window current (perhaps another window displaying the same cellview). If an equivalent window is not found, the last-created window is made current. If this window has the property `neverCurrentWindow` set to `t`, another window is made current. If no window can be made current (the CIW can never be the current window), the current window is `nil`.

Arguments

<code>w_windowId</code>	Window to close.
-------------------------	------------------

Value Returned

<code>t</code>	Returns <code>t</code> if the window is closed.
----------------	---

<code>nil</code>	Returns <code>nil</code> and issues an error message if the window is not closed.
------------------	---

hiRegCloseProc

```
hiRegCloseProc(  
    w_windowId  
    s_closeProc  
)  
=> t
```

Description

Registers the procedure to be performed when a specified window is closed.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	ID of the window for which the closing procedure is to be registered.
<i>s_closeProc</i>	Name of the procedure to be executed when the window is closed. The format for this procedure should be <code>closeProc(windowId)</code> . This closing procedure is strictly for clean up and must not call <code>hiCloseWindow</code> , any enterfunctions, or any blocking functions that display forms or dialog boxes. Because this procedure is called no matter how you close the window (from a window manager or through Cadence software), you might need to delete existing calls to clean-up functions from the user interface.

Note: A C closing procedure is called before any SKILL closing procedures. If the C procedure returns `nil`, none of the SKILL closing procedures are called and the window remains open.

Value Returned

<i>t</i>	Returns <i>t</i> if the closing procedure is registered.
----------	--

Example

```
hiRegCloseProc  
(window(2)  
'myFunction  
)=> t
```

hiUnregCloseProc

```
hiUnregCloseProc(  
    w_windowId  
    s_closeProc  
)  
=> t / nil
```

Description

Unregisters the closing procedure for a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	ID of the window associated with the closing procedure.
<i>s_closeProc</i>	Closing procedure to be unregistered.

Value Returned

<i>t</i>	Returns <i>t</i> if the closing procedure is unregistered.
<i>nil</i>	Returns <i>nil</i> if the closing procedure is not unregistered.

Example

```
hiUnregCloseProc(window(2)  
    'myFunction)
```


hiGetWindowState

```
hiGetWindowState(  
    w_windowId  
)  
=> s_state / nil
```

Description

Identifies the display state of a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to know about.
-------------------	--------------------------------

Value Returned

<i>s_state</i>	Returns <code>mapped</code> , <code>unmapped</code> , or <code>iconified</code> to describe the state of the specified window.
<code>nil</code>	Returns <code>nil</code> if the specified window is a plot window or graphics window that has never been displayed before, or if an error has occurred.

hiSwitchWindowType

```
hiSwitchWindowType(  
    w_windowId  
    t_widgetType  
    t_appType  
    t_help  
    [g_scroll]  
)  
=> t / nil
```

Description

Switches the widget type of a window from `text` to `graphics` or from `graphics` to `text`.

Other widget types are currently not supported. The application type, help string, and scroll bar option (only if switching to type `graphics`) can also be changed.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to change.
<i>t_widgetType</i>	Either <code>text</code> or <code>graphics</code> .
<i>t_appType</i>	String representing the window application type or name.
<i>t_help</i>	String used as one of the key words that Help logic uses when it is constructing the names of the various help files and/or help symbols searched for in the help files. Refer to Online Help documentation for more information. If this argument is not specified, it defaults to the same value as <i>t_applicationType</i> .
<i>g_scroll</i>	Either <code>t</code> or <code>nil</code> to specify the use of scroll bars. This function is valid only for <code>graphics</code> widget types. If set to <code>t</code> , scroll bars appear on the right and bottom of the window. If not specified or <code>nil</code> , scroll bars are not displayed.

Value Returned

`t` Returns `t` if the window type is switched.

Cadence User Interface SKILL Functions Reference

Windows

`nil` Returns `nil` and issues an error message if the window type is not switched.

Example

```
hiSwitchWindowType(window(4)  
"text" "Layout"  
"layoutHelp")  
=> t
```

hilconifyWindow

```
hiIconifyWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Converts a window into its icon.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to iconify.
-------------------	-----------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the window is iconified.
----------	--

<i>nil</i>	Returns <i>nil</i> and issues an error message if the window is not iconified.
------------	--

hiDeiconifyWindow

```
hiDeiconifyWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Converts an icon to a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to deiconify.
-------------------	-------------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the window is deiconified.
----------	--

<i>nil</i>	Returns <i>nil</i> and issues an error message if the window is not deiconified.
------------	--

hiGetWindowIconifyState

```
hiGetWindowIconifyState(  
    w_windowId  
)  
=> l_iconPosition / nil
```

Description

Returns the icon position of an iconified window.

To make sure that after restoring a saved session, the restored window is in the correct icon state, you must pass the value returned by this function as the *l_iconPosition* argument of the function [hiOpenWindow](#) or [hiCreateWindow](#) when logging either of these functions to the save/restore file.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	ID of the window for which you need the icon position.
-------------------	--

Value Returned

<i>l_iconPosition</i>	Returns the icon position (a list of the x and y coordinates) of the iconified window. The x and y coordinates correspond to the upper-left corner of the icon where 0:0 is the upper-left corner of the screen.
<i>nil</i>	Returns <i>nil</i> if the window is not iconified.

hiLowerWindow

```
hiLowerWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Lowers a window to the bottom of the list of visible windows.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to lower.
-------------------	---------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the window is lowered to the bottom of the list.
----------	--

<i>nil</i>	Returns <i>nil</i> and issues an error message if the window is not lowered to the bottom of the list.
------------	--

hiRaiseWindow

```
hiRaiseWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Raises a window to the top of the list of visible windows.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to raise.
-------------------	---------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the window is raised to the top of the list.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the window is not raised to the top of the list.

hiPickWindow

```
hiPickWindow(  
    )  
=> w_windowId / nil
```

Description

Allows selecting a window without changing the current window. When this function is called, the pointer is grabbed and not released until either the mouse button or the *Escape* key are pressed. Any other keyboard event is ignored and the mouse click is not passed on to anything else.

Value Returned

<i>w_windowId</i>	If the user clicks the mouse button while the pointer is over a window, the <i>windowId</i> for that window is returned.
<i>nil</i>	If the user selects the <i>Escape</i> key or clicks the mouse button when the pointer is not over a window, <i>nil</i> is returned.

Cadence User Interface SKILL Functions Reference

Windows

hiMapWindow

```
hiMapWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Displays a window on screen.

This window must have been displayed at one time (with [hiDisplayWindow](#) or [hiOpenWindow](#)). This function is usually used in conjunction with [hiUnmapWindow](#) (which removes a window from the screen without closing it). If a window has been iconified, [hiMapWindow](#) deiconifies the window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to map.
-------------------	-------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the window is redisplayed.
----------	--

<i>nil</i>	Returns <i>nil</i> and issues an error if <i>w_windowId</i> is invalid or if the window has never been displayed.
------------	---

hiUnmapWindow

```
hiUnmapWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Removes a window from the screen without closing it or destroying it.

This window must have been displayed at one time (with [hiDisplayWindow](#) or [hiOpenWindow](#)). This function is usually used in conjunction with [hiMapWindow](#).

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to unmap.
-------------------	---------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the window is unmapped.
----------	---

<i>nil</i>	Returns <i>nil</i> and issues an error message if <i>w_windowId</i> is invalid or if the window has never been displayed.
------------	---

Cadence User Interface SKILL Functions Reference

Windows

hiMoveWindow

```
hiMoveWindow(  
    w_windowId  
    l_point  
)  
=> t / nil
```

Description

Moves the lower left corner of a window to a new screen location.

The window manager banner is not considered the lower left corner of the window. The window, therefore, might not appear to be placed exactly at the specified point.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to move.
<i>l_point</i>	The new lower-left position of the window specified in screen coordinates. This assumes that 0:0 is the lower-left corner of the screen.

Value Returned

<i>t</i>	Returns <i>t</i> if the window is moved.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the window is not moved.

Example

```
hiMoveWindow(hiGetCurrentWindow( ) 50:100)  
=> t
```

hiResizeWindow

```
hiResizeWindow(  
    w_windowId  
    l_bBox  
)  
=> t / nil
```

Description

Resizes a window to the size of a bounding box.

The window manager banner is not considered part of this bounding box. The window, therefore, might not appear to be placed exactly at the specified points.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to resize.
<i>l_bBox</i>	The new coordinates for the bounding box of the window. Specify the lower-left and upper-right screen coordinates, assuming 0:0 to be the lower-left corner of the screen.

Value Returned

<i>t</i>	Returns <i>t</i> if the window is resized.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the window is not resized.

Examples

```
hiResizeWindow(window(4) list(0:0 500:500))  
=> t
```

You can resize window(2) to exactly match the size and position of window(1) by using the [hiGetAbsWindowScreenBBox](#) and `hiResizeWindow` functions:

```
win1BBox = hiGetAbsWindowScreenBBox(window(1) t)  
hiResizeWindow(window(2) win1BBox)
```

hiFocusToCursor

```
hiFocusToCursor(  
    g_cursorFocus  
)  
=> t
```

Description

Sets the window focus mechanism.

If *g_cursorFocus* is set to *t*, the current window is set by moving the cursor into a window. If set to *nil*, you must click in a window to make it current.

The current window is always the last window through which the cursor travels. The Command Interpreter Window can never be made current.

Note: This function is not available in Concept SKILL.

Arguments

<i>g_cursorFocus</i>	Either <i>t</i> or <i>nil</i> to determine the method of specifying the current window. If set to <i>t</i> , the current window is set by moving the cursor into a window. If set to <i>nil</i> , you must click in a window to make it current.
----------------------	--

Value Returned

<i>t</i>	Always returns <i>t</i> .
----------	---------------------------

hiGetWindowList

```
hiGetWindowList(  
    )  
=> lw_windowId
```

Description

Returns a list of all window IDs that have been created and not closed.

The Command Interpreter Window is always included in this list.

Note: This function is not available in Concept SKILL.

Arguments

None.

Value Returned

lw_windowId Returns the list of all window IDs that are currently open.

Example

```
hiGetWindowList( )  
=> (window:1 window:2 window:3)
```

hiGetWindowName

```
hiGetWindowName(  
    w_windowId  
)  
=> t_windowName / nil
```

Description

Gathers the text in the window manager frame of a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
-------------------	--

Value Returned

<i>t_windowName</i>	Returns the text contained in the window manager frame.
---------------------	---

<i>nil</i>	Returns <i>nil</i> if the window is invalid or if the window has no name associated with it (such as a plot window).
------------	--

hiSetWindowName

```
hiSetWindowName(  
    w_windowId  
    t_label  
)  
=> t / nil
```

Description

Sets the text in the window manager frame of a window.

This frame is placed around each window depending on the window manager being used.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>t_label</i>	Text to be displayed in the window manager frame.

Value Returned

<i>t</i>	Returns <i>t</i> if the text is displayed.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid.

Cadence User Interface SKILL Functions Reference

Windows

hiGetIconName

```
hiGetIconName(  
    w_windowId  
)  
=> t_iconName / nil
```

Description

Retrieves the icon name of a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
-------------------	--

Value Returned

<i>t_iconName</i>	Returns the icon name of the window specified by <i>w_windowId</i> .
-------------------	--

<i>nil</i>	Returns <i>nil</i> if the window is invalid or if the window has no name associated with it (such as a plot windows).
------------	---

Cadence User Interface SKILL Functions Reference

Windows

hiSetIconName

```
hiSetIconName(  
    w_windowId  
    t_iconName  
)  
=> t / nil
```

Description

Sets the icon name of a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>t_iconName</i>	Name to appear in the icon when the window is iconified. Only the first six or seven characters of the icon name are visible, due to the icon size.

Value Returned

<i>t</i>	Returns <i>t</i> if the icon name is set.
<i>nil</i>	Returns <i>nil</i> if the <i>w_windowId</i> is invalid.

hiSetWindowIcon

```
hiSetWindowIcon(  
    w_windowId  
    l_icon  
)  
=> t / nil
```

Description

Sets the icon of a window.

When the window is iconified, this icon is displayed by the X window manager.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>l_icon</i>	A list representing the icon. The icon can be created by <u>hiStringToIcon</u> or <u>dlDlistToIcon</u> .

Value Returned

<i>t</i>	Returns <i>t</i> if the icon is set.
<i>nil</i>	Returns <i>nil</i> if the window is invalid or if an error has occurred.

hiGetWidgetType

```
hiGetWidgetType(  
    w_windowId  
)  
=> t_widgetType / nil
```

Description

Returns the widget type of a window.

Valid widget types are `graphics`, `text`, `encap`, `browser`, `ciw`, `form`, or `none`.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
-------------------	--

Value Returned

<i>t_widgetType</i>	Returns the widget type of the window specified by <i>w_windowId</i> .
---------------------	--

<i>nil</i>	Returns <i>nil</i> if the <i>w_windowId</i> is invalid.
------------	---

hiIsWidgetType

```
hiIsWidgetType(  
    t_widgetType  
)  
=> t / nil
```

Description

Determines if a widget type is valid or not.

Note: This function is not available in Concept SKILL.

Arguments

<i>t_widgetType</i>	The widget type you want to check the validity of.
---------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the specified string is a valid widget type: graphics, text, encap, browser, ciw, form, or none.
<i>nil</i>	Returns <i>nil</i> if the <i>widgetType</i> is invalid.

Cadence User Interface SKILL Functions Reference

Windows

hiGetAppType

```
hiGetAppType(  
    w_windowId  
)  
=> t_applicationType / nil
```

Description

Returns the application type of a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
-------------------	--

Value Returned

<i>t_applicationType</i>	Returns the application type of the window specified by <i>w_windowId</i> . This type is a string set from a call to hiCreateWindow , hiOpenWindow , or <code>deOpen</code> (for example, "Schematics" or "Command Interpreter").
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiGetCurrentWindow

```
hiGetCurrentWindow(  
    )  
=> w_windowId / nil
```

Description

Returns the ID of the current window. `getCurrentWindow` is an alias for `hiGetCurrentWindow` and is defined for compatibility with earlier versions of the software.

Note: This function is not available in Concept SKILL.

Arguments

None.

Value Returned

<code>w_windowId</code>	Returns the current window identity.
<code>nil</code>	Returns <code>nil</code> if there is no current window.

hiGetMaxScreenCoords

```
hiGetMaxScreenCoords(  
    )  
=> l_coord_pair
```

Description

Retrieves the screen coordinates of the upper-right corner on the display screen. The lower-left corner of the screen always has screen coordinates 0:0.

You can use this value to determine the coordinates to specify if you want to open a window that occupies the entire screen.

Note: This function is not available in Concept SKILL.

Arguments

None.

Value Returned

l_coord_pair

Returns the screen coordinates of the upper-right corner on the display screen. These coordinates take into account the window manager banner so that a window including a window manager banner can be located at the upper-right corner of the screen.

hiGetAbsWindowScreenBBox

```
hiGetAbsWindowScreenBBox(  
    w_windowId  
    [g_includeWMOffsets]  
)  
=> l_bBox
```

Description

Returns the absolute screen coordinates of a window, assuming 0:0 is the lower-left corner of the screen.

Note: This function is not available in Concept SKILL.

Arguments

w_windowId Window where you want the function to act.

g_includeWMOffsets Either `t` or `nil` to determine if the window manager borders should be used to offset. If set to `nil` or not specified, the coordinates returned are the actual coordinates of the window area and do not include the window manager frame. If set to `t`, the coordinates returned are offset by the window manager frame size. Setting this argument to `t` allows this function to be useful when calling `hiResizeWindow()`.

Value Returned

l_bBox Returns the lower-left and upper-right screen coordinates of *w_windowId*.

Example

```
ciwBBox = hiGetAbsWindowScreenBBox(window(1))
```

hiGetWMOffsets

```
hiGetWMOffsets(  
    )  
=> l_offsets
```

Description

Returns a list of offsets (dimensions) of the window manager border placed around each window.

This function can be used to specify arguments to various window routines, like `hiGetAbsWindowScreenBBox`, when you want to take the window manager border into account.

Note: This function is not available in Concept SKILL.

Arguments

None.

Value Returned

l_offsets

Returns a four-element list of window manager banner dimensions, in the following order:

<i>x xOffset</i>	Width of the left window border in pixels.
<i>x yOffset</i>	Height of the top window border in pixels.
<i>x widthOffset</i>	Number of extra pixels added to the width of a window due to the window manager border (two times <i>x xOffset</i> in most cases).
<i>x heightOffset</i>	Number of extra pixels added to the height of a window due to the window manager border.

Cadence User Interface SKILL Functions Reference

Windows

Example

```
hiGetWMOffsets( )  
=> (6 23 12 29)
```

hiAddFixedMenu

```
hiAddFixedMenu(  
    [?window w_window]  
    ?fixedMenu r_fixMenu  
    [?menuSide s_menuSide]  
)  
=> t / nil
```

Description

Adds or replaces a fixed menu on a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window on which to add the fixed menu. This window must be of type <code>graphics</code> , <code>browser</code> , <code>form</code> , <code>text</code> , or <code>encap</code> . If you do not specify <i>w_window</i> , the current window is used.
<i>r_fixMenu</i>	Vertical fixed menu returned by a call to <code>hiCreateVerticalFixedMenu</code> . This menu can be placed on the left or right side of the window.
<i>s_menuSide</i>	Side of the window where the fixed menu is to be placed. This argument must be either <code>'left</code> , <code>'right</code> , or <code>'default</code> . If you do not specify <i>s_menuSide</i> , the default is used. The default obeys the properties set in the CIW, the User Preferences form, or the <code>.Xdefaults</code> file. To set the property in the CIW, set <code>hiGetCIWindow()->showFixedMenus</code> to either <code>left</code> or <code>right</code> . If you set this property to <code>nil</code> , the menu defaults to the left side of the window.

Value Returned

<i>t</i>	Returns <i>t</i> if the fixed menu is added.
<i>nil</i>	Returns <i>nil</i> if the fixed menu is not added.

Cadence User Interface SKILL Functions Reference

Windows

Example

```
menu = hiCreateVerticalFixedMenu(...)
hiAddFixedMenu(?window window(5) ?fixedMenu menu
?menuSide 'right)
```

hiRemoveFixedMenu

```
hiRemoveFixedMenu(  
    [w_window]  
)  
=> t / nil
```

Description

Removes the current fixed menu from a window. To delete fixed menus, first use `hiRemoveFixedMenu` to remove the fixed menu from the window, and then use [`hiDeleteMenu`](#) to destroy the fixed menu.

Note: This function is not available in Concept SKILL.

Arguments

<code>w_window</code>	Window from which the fixed menu is to be removed. If you do not specify <code>w_window</code> , the current window is used.
-----------------------	--

Value Returned

<code>t</code>	Returns <code>t</code> if the fixed menu is removed.
<code>nil</code>	Returns <code>nil</code> if no fixed menu is associated with the window.

Example

```
hiRemoveFixedMenu(window(2))
```

hiMoveFixedMenu

```
hiMoveFixedMenu(  
    [?window w_window]  
    [?menuSide s_menuSide]  
)  
=> t / nil
```

Description

Moves the fixed menu to the opposite side of the window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window with the fixed menu to be moved. If you do not specify <i>w_window</i> , the current window is used.
<i>s_menuSide</i>	Side of the window where the fixed menu is to appear. This value must be either 'left or 'right. If you do not specify a side, the menu is moved to the opposite of the current side.

Value Returned

<i>t</i>	Returns <i>t</i> if the fixed menu is moved.
<i>nil</i>	Returns <i>nil</i> if no fixed menu is associated with this window.

Example

```
hiMoveFixedMenu(?menuSide 'right)
```


hiGetWindowFixedMenu

```
hiGetWindowFixedMenu(  
    [w_window]  
)  
=> r_fixMenu / nil
```

Description

Identifies the fixed menu on a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window with the fixed menu to be identified. If you do not specify <i>w_window</i> , the current window is used.
-----------------	--

Value Returned

<i>r_fixMenu</i>	Returns the fixed menu ID.
<i>nil</i>	Returns <i>nil</i> if there is no fixed menu on the window.

Example

```
menu = hiGetWindowFixedMenu(myWin)
```

Cadence User Interface SKILL Functions Reference

Windows

hiSetCursor

```
hiSetCursor(  
    w_windowId  
    x_cursor  
)  
=> t / nil
```

Description

Sets the cursor you specify for the window.

Arguments

<i>w_windowId</i>	The window ID.
<i>x_cursor</i>	The integer Id associated with the cursor you want to set. The following table lists the constants that are legal values for this argument, as well as their equivalent X Windows cursors:

x_cursor Constant	X Windows Equivalent
hicArrow	XC_left_ptr
hicCross	XC_cross
hicHand	XC_hand2
hicHelp	XC_question_arrow
hicIbeam	XC_xterm
hicNo	XC_pirate
hicSizeAll	XC_fleur
hicSizeNESW	—
hicSizeNS	XC_sb_v_double_arrow
hicSizeNWSE	—
hicSizeWE	XC_sb_h_double_arrow
hicUpArrow	XC_sb_up_arrow
hicWait	XC_watch
hicArrowHourglass	—

Cadence User Interface SKILL Functions Reference

Windows

x_cursor Constant	X Windows Equivalent
hicHourglass	—
hicNoCursor	—

Value Returned

`t` Returns `t` if the cursor was set.

`nil` Returns `nil` if the cursor was not set.

Cadence User Interface SKILL Functions Reference

Windows

hiGetCursor

```
hiGetCursor(  
    w_windowId  
)  
=> x_cursor
```

Description

Identifies the type of cursor that is currently set for the specified window.

Arguments

<i>w_windowId</i>	The window ID.
-------------------	----------------

Value Returned

<i>x_cursor</i>	The integer ID associated with the cursor type that is currently set in the window. It could be one of the following:
-----------------	---

```
hicNoCursor  
hicArrow  
hicCross  
hicHand  
hicHelp  
hicIbeam  
hicNo  
hicSizeAll  
hicSizeNESW  
hicSizeNWSE  
hicSizeNS  
hicSizeWE  
hicUpArrow  
hicWait  
hicHourglass  
hicArrowHourglass
```

hiSetCurrentWindow

```
hiSetCurrentWindow(  
    w_windowId  
)  
=> t / nil
```

Description

Sets the current window to be the window you specify.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_windowId</i>	Window you want to make current. The Command Interpreter Window cannot be made current. If the property <code>neverCurrentWindow</code> is set to <code>t</code> (for example <code>window->neverCurrentWindow = t</code>), this window cannot be made current.
-------------------	---

Value Returned

<code>t</code>	Returns <code>t</code> if the window is made current.
<code>nil</code>	Returns <code>nil</code> if <i>w_windowId</i> is invalid, is the Command Interpreter Window, or has the property <code>neverCurrentWindow</code> set to <code>t</code> .

setCurrentWindow

The `setCurrentWindow()` function has been replaced by the [hiSetCurrentWindow](#) function.

hiSetShadowMode

```
hiSetShadowMode(  
    g_enabled  
)  
-> t / nil
```

Description

Controls the display of unselected objects in a graphics window. When you enable shadow mode with this function, all unselected objects are dimmed and all selected objects are displayed at full brightness in their original layer color instead of the selection color. You can control the level of brightness with the [hiSetShadowPercent](#) function. Note that the shadow mode setting applies to all graphics in all windows of the application, not just to the current window.

The shadow mode behavior applies only to selected objects that are drawn by Cadence applications in the selection plane. If selected objects are not drawn in the selection plane, they will not be displayed at full brightness.

Note: To use the shadow mode feature, you must turn on the selection plane. If the selection plane is not turned on, `hiSetShadowMode` dims all objects, including selected objects.

To turn on the selection plane, add the following to your `.Xdefaults` file:

```
Opus.selectionPlane: True
```

For more information about setting Cadence resources in the `.Xdefaults` file, see the [*Design Framework II Configuration Guide*](#).

Arguments

<code>g_enabled</code>	<code>t</code> or <code>nil</code> . If you specify <code>t</code> , shadow mode is enabled for all unselected objects, which means their display is dimmed. You can control the level of brightness with the hiSetShadowPercent function.
------------------------	--

Value Returned

<code>t</code>	Returns <code>t</code> if the shadow mode was enabled.
<code>nil</code>	Returns <code>nil</code> if there was an error.

hiSetShadowPercent

```
hiSetShadowPercent(  
    x_shadowPercent  
)  
-> t / nil
```

Description

Controls the level of brightness of unselected objects in a graphics window when the shadow mode is enabled. You can specify a percentage of brightness (0-100%). When the shadow percent is set to 0, the color of unselected objects is the window's background color; when the shadow percent is set to 100, the color of unselected objects is the normal layer color. Note that the shadow mode setting applies to all graphics in all windows of the application, not just to the current window. See also [hiSetShadowMode](#).

Arguments

<i>x_shadowPercent</i>	An integer between 0 and 100 which determines the percentage of brightness of unselected objects. When the shadow percent is set to 0, the color of unselected objects is the window's background color; when the shadow percent is set to 100, the color of unselected objects is the normal layer color. The default value of this argument is 50.
------------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the shadow percent was set.
<i>nil</i>	Returns <i>nil</i> if there was an error.

Viewing Functions

The user interface provides user-level functions for zooming, panning, and redrawing to ensure that all basic editing functions are done in an identical manner across all applications. Application functions created from this common core also behave similarly across the product line. Most of these functions require specific window interface routines to be registered on the windows.

Cadence User Interface SKILL Functions Reference

Windows

hiZoomIn

```
hiZoomIn(  
    [w_window]  
    [l_bBox]  
)  
=> t / nil
```

Description

Zooms in to a design.

This function also decreases the portion of the design you view.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	<p>The window you want to zoom into. You must specify the window if you specify the <i>l_bBox</i> argument. If you do not specify the <i>l_bBox</i> argument, this argument is optional.</p> <p>For the current window, you can use <code>hiGetCurrentWindow()</code> as the value of <i>w_window</i>.</p>
<i>l_bBox</i>	<p>Zoom bounding box specified in user units. If you do not specify this argument, the function prompts you for a rubberbanding box.</p> <p>After the view rectangle is specified, the command zooms in to fill the window with the contents of the view rectangle.</p>

Value Returned

<i>t</i>	Returns <i>t</i> if the magnification is increased.
<i>nil</i>	Returns <i>nil</i> if the <code>zoomPanProc</code> or <code>uPointToDbuProc</code> are not registered on the window.

Example

```
hiZoomIn(hiGetCurrentWindow() list(0:0 10:10))
```

Cadence User Interface SKILL Functions Reference

Windows

hiZoomOut

```
hiZoomOut(  
    [w_window]  
    [l_bBox]  
)  
=> t / nil
```

Description

Zoom out of a design.

This function also decreases the magnification of the design and, consequently, increases the extent of the design viewed in the window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
<i>l_bBox</i>	<p>Bounding box specified in user units. If this is not specified, the function prompts through a rubberbanding box.</p> <p>This rubberbanding box displays the aspect ratio of the window. This means the box always contains exactly what is currently viewable.</p> <p>After the rectangular area is specified, the command zooms out to fit the contents of <i>w_window</i> into the rectangular area.</p>

Value Returned

<i>t</i>	Returns <i>t</i> if the magnification is decreased.
<i>nil</i>	Returns <i>nil</i> if the <code>zoomPanProc</code> or <code>uPointToDbuProc</code> are not registered on the window.

hiZoomRelativeScale

```
hiZoomRelativeScale(  
    [w_window]  
    [n_scale]  
)  
=> t / nil
```

Description

Zooms in or out with a scale relative to the current window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
<i>n_scale</i>	Scale factor you want to use in zooming. A number greater than 1.0 specifies a zoom-in factor. A number less than 1.0 specifies a zoom-out factor. The center of the original window is the center of the new window. If <i>n_scale</i> is not specified, a scale of 1.0 is used.

Value Returned

<i>t</i>	Returns <i>t</i> if you can zoom in and out.
<i>nil</i>	Returns <i>nil</i> if the <code>zoomPanProc</code> or <code>uPointToDbuProc</code> are not registered on the window.

hiZoomAbsoluteScale

```
hiZoomAbsoluteScale(  
    [w_window]  
    [n_scale]  
)  
=> t / nil
```

Description

Zooms in or out with an absolute scale.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
<i>n_scale</i>	Scale factor you want to use for zooming. For example, a scale of 1.0 means the size of one <i>w_window</i> fits the entire representation. A scale of 2.0 means the size of two <i>w_windows</i> fits the entire representation. If <i>n_scale</i> is not specified, a scale of 1.0 is used.

Value Returned

<i>t</i>	Returns <i>t</i> if you can zoom in and out.
<i>nil</i>	Returns <i>nil</i> if the <code>zoomPanProc</code> or <code>uPointToDbuProc</code> are not registered on the window.

hiGetViewBBox

```
hiGetViewBBox(  
    [w_window]  
)  
=> l_bBox
```

Description

Returns the bounding box displayed in a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
-----------------	---

Value Returned

<i>l_bBox</i>	Returns the coordinates of the bounding box. The bounding box is a list of two points in user units describing the lower left and upper right corners of the window's view area.
---------------	--

<i>nil</i>	Returns <i>nil</i> if <i>dbuToUPProc</i> is not registered.
------------	---

Cadence User Interface SKILL Functions Reference

Windows

hiPan

```
hiPan(  
    [w_window]  
    [l_point]  
)  
=> t / nil
```

Description

Scrolls a window so that the display is centered on a point.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
<i>l_point</i>	Point you wish to be at the center of the window. If not specified, the user is prompted.

Value Returned

<i>t</i>	Returns <i>t</i> if the display is centered.
<i>nil</i>	Returns <i>nil</i> if the <code>zoomPanProc</code> , <code>dbuToUPProc</code> , or <code>uPointToDbuProc</code> are not registered on the window.

hiVectorPan

```
hiVectorPan(  
    [w_window]  
    [l_refPt]  
    [l_destPt]  
)  
=> t / nil
```

Description

Scrolls the view of the design in any direction by specifying two points in user units.

The design is panned so that *l_refPt* on the design is displayed on the screen at *l_destPt*. After the command is invoked and the first reference point is specified, a rubberbanding line connects the first point with the cursor. Click the mouse button at the desired location to specify the destination point. The design then pans to that location.

By placing the same design in two different windows and specifying the reference point in one window and the destination point in the other, the specified window can be panned relative to the area in the second window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
<i>l_refPt</i>	Reference point you want to pan from. If not specified, you are prompted to enter the value.
<i>l_destPt</i>	Destination point you want to pan to. If not specified, you are prompted to enter the value.

Value Returned

<i>t</i>	Returns <i>t</i> if the view can be scrolled as specified.
<i>nil</i>	Returns <i>nil</i> if the <i>zoomPanProc</i> or <i>uPointToDbuProc</i> is not registered on the window.

Cadence User Interface SKILL Functions Reference

Windows

hiDeltaPan

```
hiDeltaPan(  
    [w_window]  
    [n_deltax]  
    [n_deltay]  
)  
=> t / nil
```

Description

Pans the view of a design in any direction by specifying the displacements in user units.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
<i>n_deltax</i>	Number of units for panning the design in x direction. If not specified, the default of 0 is used.
<i>n_deltay</i>	Number of units for panning the design in y direction. If not specified, the default of 0 is used.

Value Returned

<i>t</i>	Returns <i>t</i> if the view can be panned in any direction.
<i>nil</i>	Returns <i>nil</i> if the <i>zoomPanProc</i> or <i>uPointToDbuProc</i> are not registered on the window.

Cadence User Interface SKILL Functions Reference

Windows

hiAbsolutePan

```
hiAbsolutePan(  
    [w_window]  
    [n_%x]  
    [n_%y]  
)  
=> t / nil
```

Description

Scrolls a window in any direction by specifying the amount to pan as a percentage of the window size.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
<i>n_%x</i>	Percentage of panning in the design in the x-direction. If not specified, the default of 0 is used.
<i>n_%y</i>	Percentage of panning in the design in the y-direction. If not specified, the default of 0 is used.

Value Returned

<i>t</i>	Returns <i>t</i> if the window can be scrolled in any direction.
<i>nil</i>	Returns <i>nil</i> if the <code>zoomPanProc</code> or <code>uPointToDbuProc</code> are not registered on the window; otherwise, returns <i>t</i> .

hiRedraw

```
hiRedraw(  
    [w_window]  
)  
=> t / nil
```

Description

Redraws the content of a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.
-----------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the content is redrawn.
----------	---

<i>nil</i>	Returns <i>nil</i> if the <code>zoomPanProc</code> is not registered on the window.
------------	---

hiSaveView

```
hiSaveView(  
    t_viewName  
    [w_window]  
)  
=> t / nil
```

Description

Saves the current viewing parameters for a window.

Note: This function is not available in Concept SKILL.

Arguments

<i>t_viewName</i>	String representing the name of the viewing parameter file.
<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.

Value Returned

<i>t</i>	Returns <i>t</i> if the parameters are saved.
<i>nil</i>	Returns <i>nil</i> if <i>w_window</i> is invalid or there is no current window.

Cadence User Interface SKILL Functions Reference

Windows

hiListView

```
hiListView(  
    [t_viewName]  
    [w_window]  
)  
=> l_viewList / nil
```

Description

Lists the saved viewing parameters for a given view, for all saved views if *viewName* is not specified.

Note: This function is not available in Concept SKILL.

Arguments

<i>t_viewName</i>	Name of the view. If a <i>viewName</i> is specified, the <i>viewList</i> returned is the bounding box of that view (equivalent to <code>hiGetViewBBox</code>). If no <i>viewName</i> is specified, the function lists the bounding boxes for each saved view. This list of lists contains the view name and the bounding box associated with each saved view.
<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.

Value Returned

<i>l_viewList</i>	Returns a list of the view name and bounding box for each view specified.
<i>nil</i>	Returns <i>nil</i> if <i>w_window</i> is invalid, if there is no current window, or if the specified <i>viewName</i> was not found.

hiRestoreView

```
hiRestoreView(  
    t_viewName  
    [w_window]  
)  
=> t / nil
```

Description

Restores a window to the view specified by *viewName*.

Note: This function is not available in Concept SKILL.

Arguments

<i>t_viewName</i>	Name of the view.
<i>w_window</i>	Window you want the function to act on. If not specified, the current window is used.

Value Returned

<i>t</i>	Returns <i>t</i> if the view is restored.
<i>nil</i>	Returns <i>nil</i> if <i>w_window</i> is invalid, if there is no current window, or if the specified <i>viewName</i> was not found.

hiPrevWinView

```
hiPrevWinView(  
    w_window  
)  
=> t / nil
```

Description

Restores a window to the view displayed before the last zoom or pan command.

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on.
-----------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the view is restored. If there was no previous zoom or pan in this window, the display is not changed and this function returns <i>t</i> .
----------	--

<i>nil</i>	Returns <i>nil</i> if <i>w_window</i> is invalid.
------------	---

hiNextWinView

```
hiNextWinView(  
    w_window  
)  
=> t / nil
```

Description

Undoes [hiPrevWinView](#).

Note: This function is not available in Concept SKILL.

Arguments

<i>w_window</i>	Window you want the function to act on.
-----------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the view is restored. If there was no previous zoom or pan in this window, the display is not changed and this function returns <i>t</i> .
----------	--

<i>nil</i>	Returns <i>nil</i> if <i>w_window</i> is invalid.
------------	---

hiGetUndoLimit

```
hiGetUndoLimit(  
    )  
=> n_undoLimit
```

Description

Returns the maximum number of [hiUndo](#) commands it is possible to perform.

This number can be changed using [hiSetUndoLimit](#).

Note: This function is not available in Concept SKILL.

Arguments

None.

Value Returned

<i>n_undoLimit</i>	Returns the maximum number of <code>hiUndo</code> commands that it is possible to perform.
--------------------	--

hiSetUndoLimit

```
hiSetUndoLimit(  
    n_undoLimit  
)  
=> t / nil
```

Description

Sets the maximum number of hiUndo commands that can be performed in sequence.

Saving of undo information is very memory intensive, so keep this number as small as possible.

Note: This function is not available in Concept SKILL.

Arguments

<i>n_undoLimit</i>	The maximum number of <code>hiUndo</code> commands that can be performed. If this number exceeds an internal maximum (currently 10), that maximum is enforced. The default value for <i>n_undoLimit</i> is 1. If set to 0, undo is disabled.
--------------------	--

Value Returned

t	Returns t if the undo limit was successfully set.
nil	Returns nil if <i>n_undoLimit</i> is non-zero and undo processing cannot be enabled.

hiUndo

```
hiUndo(  
    )  
=> t / nil
```

Description

Undoes the last user command that modified the database and that has not been undone by a previous `hiUndo`.

The number of `hiUndo` functions that can be performed in sequence is controlled by the User Preferences form or by using the [`hiSetUndoLimit`](#) function.

Not all commands that modify the database can be undone. Also note that `hiUndo` might not succeed for very large designs because of memory limitations.

`hiUndo` undoes the latest sequential database modification without regard to which window it was performed in. (In other words, if Undo is selected from the menu of window *X*, it might undo a database operation performed in window *Y*, if that was the last operation performed.)

Note: This function is not available in Concept SKILL.

Arguments

None.

Value Returned

`t` Returns `t` if an undo was successfully performed.

`nil` Returns `nil` if the last command cannot be undone.

hiRedo

```
hiRedo(  
    )  
=> t / nil
```

Description

Undoes the last [hiUndo](#), assuming no other database modifications have occurred.

hiRedo might not succeed after a successful hiUndo on very large designs because of memory limitations.

Note: This function is not available in Concept SKILL.

Arguments

None.

Value Returned

t Returns t if a redo was successfully performed.

nil Returns nil if the last command cannot be redone.

Splash Screen Functions

This section describes functions used to create splash screens and About windows.

Example of a Splash Screen



Example of an Icon



Cadence User Interface SKILL Functions Reference

Windows

hiAbout

```
hiAbout(  
    w_windowId  
)  
=> t / nil
```

Description

Displays an application's product information window, which includes the release number and copyright information.

Arguments

<i>w_windowId</i>	The window ID of the application.
-------------------	-----------------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the product information window is displayed.
----------	--

<i>nil</i>	Returns <i>t</i> if the product information window is not displayed.
------------	--

hiSetSplashBackground

```
hiSetSplashBackground(  
    t_XPMfileName  
)  
=> t /nil
```

Description

Sets the background for an application's splash screen.

See [“Example of a Splash Screen”](#) on page 509.

Arguments

<i>t_XPMfileName</i>	The name of the file containing the ASCII pixmap definition for the background.
----------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the background was set.
<i>nil</i>	Returns <i>nil</i> if the background was not set.

hiSetSplashDefaultBackground

```
hiSetSplashDefaultBackground()  
=> t /nil
```

Description

Sets the splash screen background to the default background.

See [“Example of a Splash Screen”](#) on page 509.

Value Returned

<code>t</code>	Returns <code>t</code> if the background was set to the default background.
<code>nil</code>	Returns <code>nil</code> if the background was not set to the default background.

hiSetSplashIcon

```
hiSetSplashIcon(  
    t_XPMfileName  
)  
=> t /nil
```

Description

Sets a new icon for an application's splash screen.

See [“Example of an Icon”](#) on page 509.

Arguments

<i>t_XPMfileName</i>	The name of the file containing the ASCII pixmap definition for the icon.
----------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the icon was set.
<i>nil</i>	Returns <i>nil</i> if the icon was not set.

hiSetSplashFamily

```
hiSetSplashFamily(  
    t_string  
)  
=> t /nil
```

Description

Sets the product family name on an application's splash screen.

See [“Example of a Splash Screen”](#) on page 509.

Arguments

<i>t_string</i>	The product family name.
-----------------	--------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the product family name was set.
----------	--

<i>nil</i>	Returns <i>nil</i> if the product family name was not set.
------------	--

hiSetSplashProduct

```
hiSetSplashProduct(  
    t_string  
)  
=> t /nil
```

Description

Sets the product name on an application's splash screen.

See [“Example of a Splash Screen”](#) on page 509.

Arguments

<i>t_string</i>	The product name.
-----------------	-------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the product name was set.
<i>nil</i>	Returns <i>nil</i> if the product name was not set.

hiSetSplashLicense

```
hiSetSplashLicense(  
    t_string  
)  
=> t /nil
```

Description

Sets the license information in an application's splash screen.

See [“Example of a Splash Screen”](#) on page 509.

Arguments

<i>t_string</i>	The license information you want to display in the splash screen.
-----------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the licensing information was set.
----------	--

<i>nil</i>	Returns <i>nil</i> if the licensing information was not set.
------------	--

Window Banner Functions

The following topics are discussed in this chapter:

- [Overview](#) on page 517
- [hiChangeBannerLabel](#) on page 519
- [hiDeleteBannerLabel](#) on page 520
- [hiInsertBannerMenu](#) on page 521
- [hiDeleteBannerMenu](#) on page 523
- [hiDeleteBannerMenus](#) on page 524
- [hiDeleteStatusBanner](#) on page 525
- [hiGetBannerMenus](#) on page 526
- [hiGetNumMenus](#) on page 527
- [hiIsMenuSlotFilled](#) on page 528
- [hiReplaceAllBannerMenus](#) on page 529

Overview

Labels or menus can be added to any window banner. Each window banner contains a Help button and a window number at the far right of the banner. These cannot be deleted or changed. Additional menus can be added to the window banner menu bar by calls to [hiInsertBannerMenu](#). Additional banner labels are created and inserted by a call to [hiChangeBannerLabel](#). Labels and banner menus can be deleted by calls to [hiDeleteBannerLabel](#) and [hiDeleteBannerMenu](#).

The banner menu must be created by a call to [hiCreatePulldownMenu](#) before it can be placed on a window. A maximum of four labels and 20 menus can be placed on a banner. The labels and menus are clipped if the banner is too small to display them. However, the size of the window (and its banner) can be changed using the Window Manager.

Cadence User Interface SKILL Functions Reference

Window Banner Functions

If a window has no labels or readout associated with it, the window number appears in the far right corner of the banner next to the Help button. If there are labels, a second banner is displayed above the menu banner and the window number is moved to the far right corner of this banner. Any labels appear in this second banner. If all labels are deleted, and the window does not contain any status information, this banner is also deleted.

hiChangeBannerLabel

```
hiChangeBannerLabel(  
    w_windowId  
    t_bannerLabel  
    x_labelPosition  
)  
=> t / nil
```

Description

Overwrites or adds a banner label in a window.

The labels are displayed in a second window banner above the menu bar.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>t_bannerLabel</i>	Label text.
<i>x_labelPosition</i>	Integer specifying the new position for the banner label. The positions are numbered starting from 0, beginning at the leftmost user-defined banner label. If a window label already exists at that position, the label is overwritten by this call. Otherwise, if <i>x_labelPosition</i> is greater than the current number of banner labels, the new label is added to the end (right) of the displayed labels.

Value Returned

<i>t</i>	Returns <i>t</i> upon successful modification.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the banner label is not added or overwritten.

Example

```
hiChangeBannerLabel(hiGetCurrentWindow( ) "Label 1" 0)  
=> t
```

Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiDeleteBannerLabel

```
hiDeleteBannerLabel(  
    w_windowId  
    x_labelPosition  
)  
=> t / nil
```

Description

Deletes a label in the banner of a window.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>x_labelPosition</i>	Number that indicates the label you want to delete. If you use 1 for <i>x_labelPosition</i> , the second label from the left is removed from the banner. If the label being deleted is the only label remaining on the banner and there is no status string associated with this window, the label banner is removed.

Value Returned

<i>t</i>	Returns <i>t</i> upon successful deletion.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the label is not deleted.

Example

```
hiDeleteBannerLabel(hiGetCurrentWindow( ) 0)  
=> t
```


Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiInsertBannerMenu

```
hiInsertBannerMenu(  
    w_windowId  
    g_menu  
    x_menuPosition  
)  
=> t / nil
```

Description

Inserts a menu into the banner of the window.

The *w_windowId* is returned from a call to [hiOpenWindow](#), [hiCreateWindow](#) or [hiGetCurrentWindow](#). If the menu symbol of *g_menu* already exists on the specified window, a warning message is issued and the menu is not placed on the window.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>g_menu</i>	Menu that you want to insert. The menu must be the value returned from hiCreatePulldownMenu , or a symbol representing that value. The title of this menu is displayed as text on the menu bar.
<i>x_menuPosition</i>	<p>Location in the banner where the menu is to be positioned. The menu positions are numbered from 0, starting with the leftmost user-defined menu contained within the window banner.</p> <p>This function does not overwrite a banner menu. If a menu already exists at this position, the function inserts the new menu into the specified position and moves any remaining menus to the right. If <i>x_menuPosition</i> is greater than the current number of user-defined menus, the function adds the new menu to the end (right) of all other menus but before the <i>Help</i> button.</p>

Value Returned

t Returns *t* if the menu is inserted in the banner.

Cadence User Interface SKILL Functions Reference

Window Banner Functions

`nil` Returns `nil` and issues an error message if the menu is not inserted.

Example 1

```
menu=hiCreatePulldownMenu( ... )  
hiInsertBannerMenu(window menu 0)  
=> t
```

Example 2

```
hiInsertBannerMenu(window hiCreatePulldownMenu( ... ) 0)  
=> t
```

Example 3

The following example inserts a pull-down menu in the leftmost position of the CIW. Before using this example, see the [hiCreatePulldownMenu](#) example to find out how to create the pull-down menu used in this example.

```
hiInsertBannerMenu( window(1) trPulldownMenu 0 )
```

Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiDeleteBannerMenu

```
hiDeleteBannerMenu(  
    w_windowId  
    x_menuPosition  
)  
=> t / nil
```

Description

Deletes a menu from the banner of a window.

Arguments

<i>w_windowId</i>	Window to delete menu from.
<i>x_menuPosition</i>	The menu positions are numbered from 0, starting with the leftmost user-defined menu contained within the window banner.

Value Returned

<i>t</i>	Returns <i>t</i> if the menu is deleted.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the menu is not deleted.

Example

```
hiDeleteBannerMenu(hiGetCurrentWindow( ) 0)  
=> t
```

Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiDeleteBannerMenus

```
hiDeleteBannerMenus(  
    w_windowId  
)  
=> t / nil
```

Description

Deletes all menus from the banner of a window.

Arguments

<i>w_windowId</i>	Window to delete menus from.
-------------------	------------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the menus are deleted.
----------	--

<i>nil</i>	Returns <i>nil</i> and issues an error message if the menus are not deleted.
------------	--

Example

```
hiDeleteBannerMenus(window(3))  
=> t
```

Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiDeleteStatusBanner

```
hiDeleteStatusBanner(  
    w_windowId  
)  
=> t / nil
```

Description

Deletes the status banner of a window.

This is the second optional banner that can appear above the menu banner in a window. It is used to track the cursor or to display any banner labels.

Arguments

<i>w_windowId</i>	Window from which the banner is to be deleted.
-------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the status banner is deleted.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the status banner is not deleted. If the status banner does not exist, the function returns <i>nil</i> but does not issue an error message.

Example

```
hiDeleteStatusBanner(window(3))  
=> t
```

Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiGetBannerMenus

```
hiGetBannerMenus(  
    w_windowId  
)  
=> l_menuSymbols / nil
```

Description

Creates a list of menus that are in the menu banner of a window.

Arguments

<i>w_windowId</i>	Window from which the menus are retrieved.
-------------------	--

Value Returned

<i>l_menuSymbols</i>	Returns a list of all menu symbols contained in the banner. The menu symbols are returned in the order they appear on the banner, from left to right.
<i>nil</i>	Returns <i>nil</i> if there are no menus on the banner.

Example

```
hiGetBannerMenus(hiGetCIWindow( ) )  
=> (hiOpenMenu hiDMMMenu tcMenu hiPlotMenu hiFrameworkMenu transMenu)
```

Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiGetNumMenus

```
hiGetNumMenus(  
    w_windowId  
)  
=> x_numMenus
```

Description

Counts the menus in the menu bar of a window.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
-------------------	--

Value Returned

<i>x_numMenus</i>	Returns the number of menus on the window.
-------------------	--

Example

```
hiGetNumMenus(hiGetCIWindow( ))  
=> 6
```

Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiIsMenuSlotFilled

```
hiIsMenuSlotFilled(  
    w_windowId  
    x_position  
)  
=> t / nil
```

Description

Checks a menu position in the banner of a window for the existence of a menu.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>x_position</i>	Position in the banner that you want to check. Positions are numbered the left side starting with zero.

Value Returned

<i>t</i>	Returns <i>t</i> if the position specified contains a menu.
<i>nil</i>	Returns <i>nil</i> if the position specified does not contain a menu.

Example

```
hiIsMenuSlotFilled(hiGetCIWindow( ) 5)  
=> t
```


Cadence User Interface SKILL Functions Reference

Window Banner Functions

hiReplaceAllBannerMenus

```
hiReplaceAllBannerMenus(  
    w_windowId  
    l_menulist  
)  
=> t / nil
```

Description

Replaces all banner menus in a window with new menus.

Arguments

<i>w_windowId</i>	Window where you want the function to act.
<i>l_menulist</i>	List of menus to replace the existing banner menus. These menus must be listed in the order you wish them to be placed, and each must have been created by a call to <u>hiCreatePulldownMenu</u> .

Value Returned

<i>t</i>	Returns <i>t</i> if the menus are replaced.
<i>nil</i>	Returns <i>nil</i> if the <i>w_windowId</i> or any of the menus in the list are invalid or if the same menu appears more than once in the list.

Cadence User Interface SKILL Functions Reference
Window Banner Functions

Bindkeys

This chapter discusses the following topics:

- [Overview](#) on page 532
- [hiSetBindKey](#) on page 533
- [hiSetBindKeys](#) on page 538
- [hiGetBindKey](#) on page 540
- [hiShowBindKeys](#) on page 542
- [hiShowBindKeysByAppType](#) on page 544
- [hiShowBindKeysByWindow](#) on page 546
- [hiRegisterBindKeyPrefix](#) on page 547
- [hiInheritBindKey](#) on page 548
- [hiGetBindKeyInheritRoot](#) on page 549
- [hiGetBindKeyInheritAlias](#) on page 550
- [hiGetBindKeyPrefixList](#) on page 551

Overview

This section describes how you can use SKILL functions to automate repetitive tasks by tying commands to short key strokes or key stroke combinations.

You can tie commands to single keys, key combinations, or mouse sequences. After you have defined a key or key combination to execute a certain command or a list of commands, you can use this bindkey to speed up your work. Binding keys to a task or a whole series of tasks also reduces typing mistakes.

The application name is used as the bindkey prefix and should be registered with the Design Editor via the `deRegApp` call. If you need to define the bind keys for a non-Design Editor application (one that is **not** registered through `deRegApp`, for example Tangent), the application name can be registered using the `hiRegisterBindKeyPrefix` call.

For performance reasons, a group of application names can share the same set of bind keys through bindkey inheritance. One application name can be an alias and inherit bind keys using the `hiInheritBindKey` call from another application that has bind keys defined. Currently, an alias application cannot define its own bind keys privately. It can only inherit its bindkeys from another application.

hiSetBindKey

```
hiSetBindKey(  
    [t_application_type]  
    [t_key]  
    [t_skill_cmd]  
)  
=> t / nil
```

Description

Binds a SKILL command string to a keyboard key or a mouse sequence for an application.

Before setting bindkeys, the application should be registered either through `deRegApp` or `hiRegisterBindKeyPrefix`. If `t_application_type` or `t_key` are missing, a bindkey form is displayed.

Set all bindkeys in your `.cdsinit` file or before a window for the specified application is created. For memory efficiency, do not reset bindkeys after an application window has been created.

Note: When setting multiple bindkeys at one time, the `hiSetBindKeys()` function is much more efficient than multiple calls to `hiSetBindKey()`.

The following restrictions apply to setting bindkeys:

- Do not assign any function to Control-C. This key is reserved for aborting commands. Any assignment to Control-C will be overwritten.
- Do not assign the following printable keys to the CIW or to an encapsulation window:
 - a through z, A through Z, or 0 through 9
 - ! @ # \$ % ^ & * () _ - + = | \ ~ ' { [] : ; " ' < , > . ? /

Note: If you assigned any of the above keys accidentally, the key can be restored by specifying the special case value "Xm:self-insert()" for the `t_skill_cmd`. If the key was modified in the CIW, use the bindkey form to restore the behavior, since you will not be able to type the key you are trying to restore in the CIW anymore.

- Do not use the left or middle mouse buttons for setting bindings in the CIW and encapsulation windows. Setting bindings for these buttons conflicts with their normal actions.
- Do not assign anything to F3 or F4 function keys. Cadence software uses the F3 key to display options forms and the F4 key for partial selection in graphics windows.

Cadence User Interface SKILL Functions Reference

Bindkeys

- Do not assign anything to the following keys, depending on your window manager:

Motif and OpenLook window managers use the F1 key for help and the F10 key for highlighting menus. F1 key is usually only used for Help if no explicit Help key exists on the keyboard.
- SunOS reserves the F11 and F12 function keys, and they cannot be overridden. This restriction might apply to other platforms as well.
- The unmodified F10 function key is used for `osfMenuBar` and cannot be overridden on SunOS. This restriction might apply to other platforms as well.

Arguments

t_application_type

Any user-definable string that has been registered through `hiRegisterBindKeyPrefix`. If not specified, a bindkey form is displayed. The application type could be one of the following strings:

Application Type	Meaning
Command Interpreter	Command Interpreter Window
Schematics	Schematics Window
Show File	Show File Window
Layout	Layout Window

t_key

String specifying the key sequence, the operator, detail, and occurrences. If not specified, a bindkey form is displayed. Items in this list are

modifier_list (optional)
operator
detail (optional)
occurrences (optional)

If *t_key* ends with “EF”, the SKILL command is used in `enterfunction` mode. Otherwise, it is a non-`enterfunction` mode command. If there is no `enterfunction` mode command defined when a key or mouse

Cadence User Interface SKILL Functions Reference

Bindkeys

event happens in `enterfunction` mode, the non-`enterfunction` mode command for this key is used.

Note that even an empty string is a valid bindkey command. Therefore, if you want a non-`enterfunction` mode command to be executed during an `enterfunction`, there should not be an `enterfunction` mode command defined for this key.

You would use `t_key` in the following syntax:

```
"modifier_list<operator>detail(occurrences)"
```

Legal modifiers

Ctrl	Control modifier bit
Shift	Shift modifier bit
Meta	Meta key modifier
Hyper	Hyper key modifier
Super	Super key modifier
Alt	Alt key modifier
None	No modifiers

Note: All modifiers except Ctrl and Shift will be checked to ensure they are assigned to keys on the current keyboard. A warning message will be issued, and the bindkey will not be defined for any modifier that is not assigned to a keyboard key. The Ctrl and Shift modifiers are guaranteed to be assigned. The Caps Lock ("Lock") key cannot be specified as a modifier and the state of the Caps Lock will always be ignored for Opus bindkeys.

It is not legal to use an exclamation point or a tilde in a bindkey string.

If `None` is specified, it means no modifiers can be asserted. This is also the equivalent of having no modifiers.

None <event> detail

Cadence User Interface SKILL Functions Reference

Bindkeys

<event> detail

Supported operators

Btn1Down	Button 1 Press
Btn2Down	Button 2 Press
Btn3Down	Button 3 Press
Key	Key Press
DrawThru1	Button 1 Press and Motion
DrawThru2	Button 2 Press and Motion
DrawThru3	Button 3 Press and Motion

Examples:

For single click on Button 1 Down with Shift and Meta, use this specification:

```
"Shift Meta<Btn1Down>"
```

For double-click on Button 1 Down with Shift, use this specification:

```
"Shift<Btn1Down>(2)"
```

A double click event is detected when the mouse button is pressed twice rapidly in succession without moving the mouse pointer (namely, button down, button up, button down). The time interval between the two button down events must be within a specified double click time. (This interval defaults to 200 milliseconds, but it can be reset from the User Preferences Form of the CIW Utilities Menu.)

For drawthru on button 3, use this specification:

```
"<DrawThru3>"
```

Drawthru is detected by pressing the mouse button and dragging the mouse before releasing the button. This operator should be specified only for graphics windows.

The following are examples of various key bindings:

```
"Shift<Key>F2" (shift and the F2 function key)
```

```
"Ctrl Shift<Key>Escape" (control, shift, and escape keys)
```

```
"<Key>3" (3 key)
```

When setting a bindkey to the comma key, spell out the word comma rather than using the comma character (,).

Cadence User Interface SKILL Functions Reference

Bindkeys

"<Key>comma"
"Shift<Key>comma"

t_skill_cmd

SKILL function to be executed with the translation in *t_key*. If you wish to specify more than one function, separate each one with a space. If a function is not specified, a bindkey form is displayed.

The maximum string length for the SKILL command is 511 characters. If multiple commands are to be executed, it is recommended to define a SKILL procedure and call the SKILL procedure rather than coding the entire procedure into the bindkey command string.

Note: To unset a previously set bindkey, you can set the *t_skill_cmd* to "". This does not remove the bindkey, but causes an empty SKILL function to be executed.

Value Returned

t

Returns *t* if the bindkey settings are made.

nil

Issues a warning and returns *nil* if *t_application_type* or *t_key* is invalid or if the *t_skill_cmd* string is longer than 511 characters.

Examples

```
hiSetBindKey("Command Interpreter" "None<Key>F6"  
"hiQuit( )")  
=> t  
  
hiSetBindKey("Schematics" "Shift<Key>z"  
"hiZoomIn(hiGetCurrentWindow( ))")  
=> t  
  
hiSetBindKey("Schematics" "Shift Ctrl<Key>s"  
"{println(\"Editing selected object \")  
hiEditPropList(car(geGetSelectedSet( )))}")  
=> t  
  
hiSetBindKey("Schematics" "None <Btn1Down> EF"  
"mouseAddPt( )")  
=> t  
  
hiSetBindKey("Schematics" "<DrawThru3>" "hiZoomIn( )")  
=> t  
  
hiSetBindKey("Show File" "<Key>F8"  
"hiUnselectTextAll(hiGetCurrentWindow( ))")  
=> t
```

Cadence User Interface SKILL Functions Reference

Bindkeys

hiSetBindKeys

```
hiSetBindKeys(  
    t_application_type  
    l_bindKeyList  
)  
=> t / nil
```

Description

Sets multiple bindkeys for an application at one time. For more information about restrictions, see [_hiSetBindKey\(\)](#).

Arguments

t_application_type

Any user-definable string that has been registered through [_hiRegisterBindKeyPrefix](#). The application type could be one of the following strings:

Application Type	Meaning
Command Interpreter	Command Interpreter Window
Schematics	Schematics Window
Show File	Show File Window
Layout	Layout Window

l_bindKeyList

The list of bindkey lists. Each list is a bindkey setting with the following format:

```
list(  
    t_key  
    t_skill_cmd  
    [t_EFskill_cmd]  
)
```

For information about *t_key* and *t_skill_cmd*, see the [_hiSetBindKey](#) function. *t_EFskill_cmd* is the same as *t_skill_cmd* when the *t_key* argument ends with "EF" (enterfunction mode). Do NOT add EF to the end of the *t_key* argument.

Cadence User Interface SKILL Functions Reference

Bindkeys

Note: The only way to invalidate a command is to replace it with another command or with an empty string. If you set the *t_skill_cmd* argument to `nil` when a non-enterfunction command is already defined for the key, it will not be changed. This rule also applies to *t_EFskill_cmd* and enterfunction commands if they are `nil` or if they are not passed.

Cadence User Interface SKILL Functions Reference

Bindkeys

hiGetBindKey

```
hiGetBindKey(  
    [t_application_type]  
    [t_key]  
)  
=> t_skill_cmd / nil
```

Description

Returns the SKILL command string bound to a key or mouse button for an application.

If *t_application_type* or *t_key* is not specified, a bindkey form is displayed.

Arguments

t_application_type

Any user-type string that has been registered using `deRegApp` or `hiRegisterBindKeyPrefix`. If not specified, a bindkey form is displayed. The application type can be a string like one of the following:

Command Interpreter	Command Interpreter Window (CIW)
Schematics	Schematics Window
Show File	Show File Window
wa	Waveform Window

t_key

String specifying the key sequence, the operator, detail, and occurrences. If not specified, a bindkey form is displayed. Items in this list are

modifier_list (optional)
operator
detail (optional)
occurrences (optional)

If *t_key* ends with “EF”, the SKILL function is used in `enterfunction` mode. If not, it is a `non-enterfunction` mode command.

Cadence User Interface SKILL Functions Reference

Bindkeys

Value Returned

<i>t_skill_cmd</i>	Returns the associated SKILL function string if one is associated with the specified application type and key sequence.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>t_application_type</i> or <i>s_key</i> is invalid.

Cadence User Interface SKILL Functions Reference

Bindkeys

hiShowBindKeys

```
hiShowBindKeys(  
    [g_appTypeOrWindow]  
)  
=> t / nil
```

Description

Displays the bindkeys for an application type or window in a Viewfile window.

You can save the window contents to a file by selecting *Save* or *Save As* from the *File* pulldown menu or using the `hiSaveViewfile()` or `hiSaveAsViewfile()` functions. After saving the file, you can load it into another Opus session using the `load()` or `loadi()` SKILL functions.

If there is any bindkey inheritance for this bindkey table, the application aliases are displayed at the top of the window. If `g_appTypeOrWindow` is missing, a bindkey form is displayed.

This is a wrapper function. Depending on the type of the argument, it calls `hiShowBindKeysByAppType` or `hiShowBindKeysByWindow`.

Arguments

`g_appTypeOrWindow` String or window ID. If it is a string, the argument is interpreted as an application type and the bind keys are displayed using the `hiShowBindKeysByAppType` function. If it is a window ID, the bind keys are displayed using the `hiShowBindKeysByWindow` function.

If the application type is specified, it should be previously registered either through `deRegApp` or `hiRegisterBindKeyPrefix` and can be one of the following:

Command Interpreter	Command Interpreter Window (CIW)
Schematics	Schematics Window
Show File	Show File Window
wa	Waveform Window

Cadence User Interface SKILL Functions Reference

Bindkeys

Value Returned

<code>t</code>	Returns <code>t</code> if the bindkey is displayed.
<code>nil</code>	Issues a warning and returns <code>nil</code> if <i>g_appTypeOrWindow</i> is invalid.

Cadence User Interface SKILL Functions Reference

Bindkeys

hiShowBindKeysByAppType

```
hiShowBindKeysByAppType(  
    w_window1  
    t_application_type  
)  
=> t / nil
```

Description

Displays the bindkey for an application in a Viewfile window.

You can save the window contents to a file by selecting *Save* or *Save As* from the *File* pulldown menu or using the `hiSaveViewfile` or `hiSaveAsViewfile` functions. After saving the file, you can load it into another Opus session using the `load()` or `loadi()` SKILL functions.

If there is any bindkey inheritance for this bindkey table, the application aliases are displayed at the top of the window.

Arguments

w_window1 The window in which you want to display the bindkeys.

t_application_type Any user-defined application type registered with *deRegApp* or *hiRegisterBindKeyPrefix*. It can be one of the following:

Command Interpreter	Command Interpreter Window (CIW)
Schematics	Schematics Window
Show File	Show File Window
wa	Waveform Window

Note: *t_application_type* can be any user-definable string.

Value Returned

t Returns *t* if the bindkey is displayed.

Cadence User Interface SKILL Functions Reference

Bindkeys

<code>nil</code>	Issues a warning and returns <code>nil</code> if <i>t_application_type</i> is invalid.
------------------	--

Cadence User Interface SKILL Functions Reference

Bindkeys

hiShowBindKeysByWindow

```
hiShowBindKeysByWindow(  
    [w_window1]  
    [w_window2]  
)  
=> t / nil
```

Description

Displays the bindkeys for a window in a Viewfile window.

You can save the window contents to a file by selecting *Save* or *Save As* from the *File* pulldown menu or using the `hiSaveViewfile()` or `hiSaveAsViewfile()` functions. After saving the file, you can load it into another Opus session using the `load()` or `loadi()` SKILL functions.

If there is any bindkey inheritance for this bindkey table, the application aliases are displayed at the top of the window.

Arguments

<code>w_window1</code>	Text window in which you want to display the bindkeys.
<code>w_window2</code>	Window associated with the application for which you want to see the bindkeys.

Value Returned

<code>t</code>	Returns <code>t</code> if the bindkey is displayed.
<code>nil</code>	Issues a warning and returns <code>nil</code> if <code>w_window</code> is invalid.

Cadence User Interface SKILL Functions Reference

Bindkeys

hiRegisterBindKeyPrefix

```
hiRegisterBindKeyPrefix(  
    t_application_type  
    [t_widgetType]  
)  
=> t / nil
```

Description

Registers the application bindkey prefix that you do not want or need to register with the Design Editor application.

This function should be called only when you want to have bindkeys for the application. The Design Editor application should be registered via the `deRegApp` call that calls this routine.

Arguments

t_application_type

User-defined application type registered either through `deRegApp` or `hiRegisterBindKeyPrefix`. It can be one of the following:

Command Interpreter	Command Interpreter Window (CIW)
Schematics	Schematics Window
Show File	Show File Window
wa	Waveform Window

t_widgetType

Widget type of the window with which the bindkey table is associated. The string *t_widgetType* corresponds to the argument passed to `hiCreateWindow`.

Value Returned

t

Returns *t* if the bindkey prefix is registered.

nil

Issues a warning and returns nil if *t_application_type* has already been registered.

Cadence User Interface SKILL Functions Reference

Bindkeys

hiInheritBindKey

```
hiInheritBindKey(  
    t_application_type  
    t_from_app_type  
)  
=> t / nil
```

Description

Causes a registered application to inherit bindkeys from another application.

Nested inheritance is not allowed, that is, *t_from_app_type* cannot inherit bindkeys from another application.

Bindkeys cannot be defined for an application that inherits bindkeys. Also, an application can only inherit from one application and multiple inheritance is not allowed. However, any number of applications can inherit from any particular application.

Arguments

t_application_type

Application type (previously registered using *deRegApp* or *hiRegisterBindKeyPrefix*) to inherit the bindkeys.

t_from_app_type

Application type (previously registered using *deRegApp* or *hiRegisterBindKeyPrefix*) that has or will have bindkeys defined.

Value Returned

t

Returns *t* if the bindkeys are inherited.

nil

Issues a warning and returns nil if *t_application_type* or *t_from_app_type* is invalid, if *t_application_type* already has bindkeys defined, or if *t_from_app_type* inherited its bindkeys from another application.

Cadence User Interface SKILL Functions Reference

Bindkeys

hiGetBindKeyInheritRoot

```
hiGetBindKeyInheritRoot(  
    t_application_type  
)  
=> t_application / nil
```

Description

Returns the application name from which *t_application_type* inherits its bindkeys.

If *t_application_type* does not inherit bindkeys from another application, returns *t_application_type*.

Arguments

t_application_type

Application type that does not define bindkeys but inherits them from another application.

Value Returned

t_application

Returns the name of the application from which the bindkeys were inherited.

nil

Issues a warning and returns nil if *t_application_type* is invalid.

Cadence User Interface SKILL Functions Reference

Bindkeys

hiGetBindKeyInheritAlias

```
hiGetBindKeyInheritAlias(  
    t_application_type  
)  
=> l_application / nil
```

Description

Returns a list of application names that inherit the bindkeys from *t_application_type*.

Arguments

<i>t_application_type</i>	Application type that defines bindkeys to be inherited by another application.
---------------------------	--

Value Returned

<i>l_application</i>	Returns a list of applications that inherit bindkeys from this application type.
<i>nil</i>	Issues a warning and returns <i>nil</i> if there are no applications that inherit bindkeys from the specified application type, or if <i>t_application_type</i> is invalid.

hiGetBindKeyPrefixList

```
hiGetBindKeyPrefixList(  
    )  
=> l_application
```

Description

Returns a list of registered applications. This list is used to build the menu for the Cyclic button on the Bind Keys form.

Value Returned

l_application

Returns the list that is used to build the menu for the Cyclic button on the Bind Keys form. Root applications are listed with no leading spaces. Sub-applications (applications that inherit from a root application) are listed following the root application from which they inherit and have three leading spaces in the string.

Note: Root applications are registered using [hiRegisterBindKeyPrefix](#), and sub-applications are registered using [hiInheritBindKey](#).

Cadence User Interface SKILL Functions Reference
Bindkeys

User Entry Functions

The following topics are discussed in this chapter:

- Overview on page 554
 - ❑ Origin of Enterfunction Data on page 554
 - ❑ Enterfunction Prompts on page 555
 - ❑ Option Forms on page 555
 - ❑ Preloading Points on page 556
 - ❑ Number of Points To Get on page 556
 - ❑ Terminating an Enterfunction on page 556
 - ❑ Callback Procedures on page 557
 - ❑ Enterfunction Flags on page 558
 - ❑ Nesting Enterfunctions on page 559
- User Entry Functions on page 562

Overview

User entry functions collect data (such as the points of a shape, a string, or a number) that you provide. These functions are known as “enterfunctions” and this term is used throughout this section.

Enterfunctions are a special class of SKILL functions specific to a window or group of windows. They let you digitize shapes or enter a string or number. When you invoke an enterfunction, the current window is the only window where data can be entered. Some applications, such as the layout editors, have a group of windows where each of the windows is viewing the same database cell. In this case, the enterfunctions let you enter data in any of the windows. Data entered in a window that is not in the window group has no effect on an enterfunction.

The functions are nestable, meaning that you can be entering a box and then start entering an arc. When you finish digitizing the arc, you can complete the box.

Because enterfunctions are specific to a window or group of windows, you can have separate enterfunctions active in unrelated windows. To illustrate this, suppose you have two unrelated windows A and B. You start an `enterBox` function in window A and then start an `enterLine` function in window B. You can enter a point for the box in window A, enter a line point on window B, and so forth. The system treats them as two separate functions.

The `enterMultiRep` enterfunction is an exception to this rule and is described later in this section.

If an enterfunction is used inside an application window, any adjustments to the list of points and/or “snapping” points is performed by the application. Please refer to the documentation for the specific application for more information.

Origin of Enterfunction Data

Input to enterfunctions can be entered in two ways:

- You type in the data.
- A SKILL procedure adds data to the enterfunction.

If you type in data, it is added to the enterfunction. You can type it directly or invoke a function that returns the data.

Usually, a function that adds data to an enterfunction is bound to a mouse button or key with the `hiSetBindKey` function.

Cadence User Interface SKILL Functions Reference

User Entry Functions

Enterfunction Prompts

Prompts let you know that you have an enterfunction active in a window. A prompt string (supplied to the enterfunction) is displayed on the CIW prompt line whenever you enter data into a window that has an enterfunction active. Prompt strings should be informative and let users know what the state of the enterfunction is, as well as what they should enter.

If an enterfunction requires multiple points, the caller supplies a list of prompt strings, one for each point. The prompt displayed in the CIW advances one item in the list of strings as each point is entered. Suppose that an enterfunction requires two points and the prompt list provided is

```
('("Enter First Point" "Enter Second Point"))
```

Before you enter a point, the prompt line would say

```
"Enter First Point"
```

After you have entered the first point, the next string is displayed

```
"Enter Second Point"
```

If you delete a point, the previous prompt message is displayed. If the list of prompt strings is exhausted before you enter all the points, the last prompt message continues to display. Extra prompt strings are ignored. If a prompt list is not supplied, prompts are not displayed. (This practice is not recommended.)

Option Forms

An enterfunction can also have an “option form” associated with it. It allows the user to enter information or options associated with the enterfunction. Option forms are created using [hiCreateOptionsForm](#).

The options form can be made visible or invisible at any time by calling the SKILL function [hiToggleEnterForm](#). When the enterfunction ends, the options form is automatically removed from the screen. When you are in novice mode, option forms are displayed automatically as soon as the enterfunction starts. In other modes, you must explicitly bring the form up with [hiToggleEnterForm](#).

To get immediate response to any changes the user makes to the values on option forms, call the [changeEnterFun](#) function.

Preloading Points

There are times when you might know some of the points of a shape, and need further information to complete it. You can pass the enterfunction a list of points to preload before digitizing the shapes. Note that if prompt strings are supplied, those corresponding to the preloaded points are skipped, and the prompt corresponding to the first digitized point is the first prompt displayed.

All points entered with the mouse are automatically adjusted to the current snap grid and use the current snap shape mode and rule. Preloaded points are also adjusted in this manner. For enterfunctions where they exist, collinear points are automatically removed.

Number of Points To Get

Sometimes you might need to enter a limited number of points. For example, you might need a line that has four points. You can tell the enterfunction to stop after it has obtained four points.

Terminating an Enterfunction

Enterfunctions are terminated in one of several ways. All enterfunctions can be cancelled, by calling `cancelEnterFun`, or finished, by calling `finishEnterFun`. When an enterfunction is cancelled, all data entered is lost and `nil` is returned. When it is finished, the data entered is returned as the result of the enterfunction.



Do NOT use the return value of the enterfunction to gather data but to register a callback procedure for this purpose. Enterfunctions will no longer block in future releases, and the return value will indicate only that the enterfunction was successfully started.

For functions that take a specific number of points, such as `enterArc` or `enterBox`, the enterfunction is automatically terminated when the last point is entered. For functions taking a variable number of points, termination occurs automatically only when two identical points are entered. Otherwise, termination occurs only with an explicit call to `finishEnterFun` or `cancelEnterFun`. Note that if a pointlist is passed for preloading to an enterfunction that requires a variable number of points, the enterfunction returns without further interaction if the last two points in the list are identical.

Callback Procedures

Enterfunctions do nothing but gather data. As such, they are service layer functions and are used by higher level procedures, such as adding a polygon to a database cell.

To allow the higher level procedures to respond to or to monitor the progress of the enterfunction, callback procedures should be supplied to the enterfunction. These procedures are not required, but allow an application to trap a condition as soon as the condition becomes known. This prevents having to enter twenty points of a shape, only to find that the second point caused a problem. The enterfunctions can have registered callback procedures that are invoked for the following events:

- The enterfunction is starting
- A point is entered
- A point is deleted
- The enterfunction is finished or cancelled

These functions are known as the `initProc`, `addPointProc`, `delPointProc`, `formProc`, and `doneProc`.

Callback Procedure `initProc`

`initProc` is called as soon as the enterfunction starts and is passed the `w_windowId`. `initProc` is called before any data entry is done and should initialize any data that might be needed during data entry.

```
initProc( w_windowId )
```

Callback Procedure `addPointProc`

`addPointProc` is called after you enter a point and is passed the `w_windowId` and the list of points that you have entered so far. The last point in the list is the point you just entered.

```
addPointProc( w_windowId l_pointList )
```

Note: To reject the most recently added point, `addPointProc` should call the `deletePoint` function.

Cadence User Interface SKILL Functions Reference

User Entry Functions

Callback Procedure delPointProc

`delPointProc` is called when you delete a point and is passed the `w_windowId` and the list of points you have entered so far. The last point in the list is the point that is about to be removed.

```
delPointProc( w_windowId l_pointList )
```

Callback Procedure formProc

`formProc` is called when the options form is about to be displayed. It is passed the `w_windowId`, the symbol for the options form, and a flag indicating whether the form is being displayed (`g_map == t`) or erased (`g_map == nil`). You normally use this to make sure the form is initialized with current values.

```
formProc( w_windowId s_form g_map )
```

Callback Procedure doneProc

`doneProc` is called when you finish an enterfunction or cancel it. It is passed the `w_windowId`, either `t` or `nil` (`t` if the enterfunction completed, `nil` if it was cancelled), and the list of points that you entered. For `enterString`, the string that you typed is passed instead of a point list. For `enterNumber`, the number you typed is passed instead of a point list. `doneProc` is very important when enterfunctions are nested. `doneProc` and nesting are discussed at the end of this section.

```
doneProc( w_windowId t | nil l_pointList )
```

Enterfunction Flags

All enterfunctions can have certain flags passed that control the operation of the enterfunction. These flags control such things as

- Drawing rubber band lines
- Automatic display of the options form
- Acceptance of a string or number without termination
- The use of infix

Enterfunction Flag dontDraw

If `dontDraw` is `nil` (the default), the rubber band lines are drawn between the entered points. Otherwise, these lines are not displayed.

Cadence User Interface SKILL Functions Reference

User Entry Functions

Enterfunction Flag `alwaysMap`

If `alwaysMap` is `nil` (the default), the options form is displayed only if you request it with `hiToggleEnterForm` or if you are in novice mode. Otherwise, the options form is displayed automatically when the enterfunction is called.

Enterfunction Flag `acceptString`

If `acceptString` is `nil` (the default) and you enter a string into the CIW, the enterfunction is terminated immediately. Otherwise, you can enter the string without terminating the enterfunction. In the latter case, `doneProc` is called with the string instead of the entered points. `doneProc` knows that the enterfunction is not really being terminated.

Enterfunction Flag `acceptNumber`

If `acceptNumber` is `nil` (the default) and you enter a number into the CIW, the enterfunction is terminated immediately. Otherwise, you can enter the number without terminating the enterfunction. In this latter case, `doneProc` is called with the number instead of the entered points. `doneProc` knows that the enterfunction is not really being terminated.

Enterfunction Flag `noInfix`

Infix mode allows you to specify the first point for any operation implicitly. When infix is on and an enterfunction is executed through a bindkey in some window, the cursor location at which the function was executed is taken as the first point entered in the enterfunction. If infix is off, this point is ignored and you must explicitly enter it. Infix cannot be used from a menu item because there is no location implicitly specified. The `noInfix` flag, when `nil` (the default), allows infix to be used when the global infix flag (`window(1)->infixOn`) is set and when the function is called from a bindkey. If `noInfix` is not `nil`, then infix is not used no matter what the setting of the global flag.

Nesting Enterfunctions

Most enterfunctions can be nested inside one another, meaning that while one is active, another can be started. When the nested enterfunction terminates, the previous one again becomes active, and its state at the time of the second enterfunction is restored.

Because you can nest enterfunctions, and because the functions block until they are cancelled or completed, it helps to be aware of what happens inside the software. Enterfunctions start new SKILL top levels, in which the flow of control is passed from one `topLevel` to another. The processing of user events does not stop, but evaluation of

Cadence User Interface SKILL Functions Reference

User Entry Functions

statements after the `enterfunction` is blocked until the `topLevel` is completed (either by completing an `enterfunction` or by cancelling it).

Suppose you have two procedures A and B

```
procedure( A( )
  prog( (result)
    result = enterBox( )
    println("The result from A is %L\n" result)
    return(result)
  )
)

procedure( B( )
  prog( (result)
    result = enterArc( )
    println("The result from B is %L\n" result)
    return(result)
  )
)
```

and you have two windows, `window1` and `window2`. If `window1` is current and you type `A()` in the CIW, procedure A executes. From within procedure A, `enterBox` is called and evaluation blocks at this statement, waiting for you to complete the `enterfunction`.

Now suppose you went to `window2`, made it current, and then invoked procedure B. Procedure B calls `enterArc` and evaluation blocks at this statement, waiting for you to complete the arc.

The system allows you to enter points in either `window1` or `window2`. Suppose you enter the two points of the box in `window1`. The `println` will not be executed because evaluation is blocked inside procedure B. As soon as you finish the arc in `window2`, the `println` statement in procedure B is evaluated. Procedure B then returns, allowing procedure A to return, and its `println` statement is evaluated.

Due to this blocking behavior, we strongly suggest that you use a `doneProc` whenever you need to do something immediately after you complete an `enterfunction`.

If we rewrote the previous example using a `doneProc`, it would behave better.

```
procedure( resultsA( windowId doneOrCancel pointsList)
  if( doneOrCancel == t) /* it was finished */
  then println("The result from A is %L" pointsList)
)

procedure( A( )
  enterBox(?doneProc "resultsA")
)

procedure( resultsB( windowId doneOrCancel pointsList)
  if( doneOrCancel == t) /* it was finished */
  then println("The result from B is %L" pointsList)
)
```


Cadence User Interface SKILL Functions Reference

User Entry Functions

```
    )  
  )  
procedure( B( )  
    enterArc(?doneProc "resultsB")  
)
```

The trick to using the enterfunctions in a nested environment is to organize your code in a callback style, doing what you want to do with the data in a callback procedure, rather than after the enterfunction statement.

User Entry Functions

This section describes the following entry functions:

- [enterArc](#) on page 564
- [enterBox](#) on page 567
- [enterCircle](#) on page 571
- [enterDonut](#) on page 574
- [enterEllipse](#) on page 578
- [enterLine](#) on page 581
- [enterNumber](#) on page 584
- [enterPath](#) on page 586
- [enterPoint](#) on page 590
- [enterPoints](#) on page 593
- [enterPolygon](#) on page 596
- [enterScreenBox](#) on page 599
- [enterSegment](#) on page 601
- [enterMultiRep](#) on page 604
- [enterString](#) on page 608
- [addPoint](#) on page 610
- [preXY](#) on page 612
- [deletePoint](#) on page 613
- [cancelEnterFun](#) on page 615
- [finishEnterFun](#) on page 616
- [applyEnterFun](#) on page 617
- [changeEnterFun](#) on page 619
- [changeNextEnterFun](#) on page 622
- [clearAllEnterFunctions](#) on page 623

Cadence User Interface SKILL Functions Reference

User Entry Functions

- [hiGetCurrentCmd](#) on page 624
- [hiInEnterFun](#) on page 625
- [hiMarkNonNestable](#) on page 627
- [hiUpdate](#) on page 628
- [hiToggleEnterForm](#) on page 629
- [undrawEnterFun](#) on page 630
- [drawEnterFun](#) on page 631

Cadence User Interface SKILL Functions Reference

User Entry Functions

enterArc

```
enterArc(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_point_list / nil
```

Description

Enters an arc into the current window.

An arc is defined as a portion of a horizontally or vertically oriented ellipse. It is specified by three points: the starting point of the arc on the ellipse, the ending point of the arc on the ellipse, and a point between them through which the arc must pass. An arc can subtend no more than 180 degrees.

Arguments

<i>l_promptList</i>	List of prompt strings to appear on the CIW prompt line. If supplied, only the first three strings are used. Each string corresponds to one of the three points needed to specify the arc.
---------------------	--

<i>l_pointList</i>	List of no more than three points specifying the arc. If all three points are supplied, the enterfunction returns the list after adjustment. Otherwise, the supplied points are adjusted and displayed as if you had entered them explicitly.
--------------------	---

After the first point is entered, a rubberband line is drawn between this point and the cursor's current position. After the second point is entered, a rubberbanding arc is drawn so that it passes through the first and second points and the cursor's

Cadence User Interface SKILL Functions Reference

User Entry Functions

current position. When the third point is entered, the function returns the point list.

If you attempt to enter a point through which a legal arc cannot be drawn, nothing is displayed. If the function is terminated at this point, no points are returned.

<i>s_form</i>	Options form created with <u>hiCreateOptionsForm</u> . This form is displayed when you call <u>hiToggleEnterForm</u> . It contains options relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubberband lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Cadence User Interface SKILL Functions Reference

User Entry Functions

Value Returned

<code>l_point_list</code>	Returns the list of points you entered in user units.
<code>nil</code>	Returns <code>nil</code> if the function is cancelled or if an arc could not be generated from the entered points.

Note: Retrieve data through the callback procedure `t_doneProcName` instead of using the return value for this enterfunction.

Example

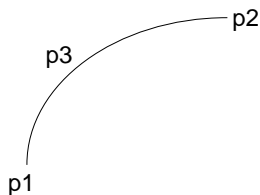
```
procedure( arcDone( w done pts )
  if( done then
    printf("Arc points are %L.\n" pts)
  else
    println("Arc entry aborted.")
  )
)
enterArc( ?prompts list( "Enter the first point of your arc."
  "Enter the second point of your arc."
  "Enter the last point of your arc." )
  ?doneProc "arcDone"
)
```

The prompt “Enter the first point of your arc” is displayed on the CIW prompt line. When you enter point *p1*, the prompt changes to “Enter the second point of your arc” and a rubber band line appears from *p1* to the cursor. Enter point *p2* and the line becomes an arc while the prompt changes to “Enter the last point of your arc.” Finally, enter point *p3* and `doneProc` prints the three arc points:

Arc points are (*p1 p2 p3*).

The enterfunction returns a list of the three arc points.

(*p1 p2 p3*)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterBox

```
enterBox(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_bBox / nil
```

Description

Enters a box in the current window.

Arguments

<i>l_promptList</i>	List of prompt strings to appear on the CIW prompt line. If supplied, only the first two strings are used. Each string corresponds to one of the two points needed to specify the box.
<i>l_pointList</i>	List of no more than two points specifying the box. If both points are specified, the box is returned after adjustment. If not, you are prompted for the point or points not specified. If one point is supplied, it is adjusted as if you had entered it. After you enter the first point, a rubberbanding box is drawn between the first point and the cursor's current position. After the second point is entered, the function returns a list of two points. The list returned is normalized so that the first point corresponds to the lower left corner of the box; the second is the upper right corner.
<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Value Returned

<i>l_box</i>	Returns a list of two points.
<i>nil</i>	Returns <code>nil</code> if the enterfunction is cancelled.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( boxDone( w done pts )
  if( done then
    printf("Box entered was %L. Layer is %L %s.\n" pts
      list( myForm->layer->value
```


Cadence User Interface SKILL Functions Reference

User Entry Functions

```
        myForm->purpose->value)
    hiGetWindowName(w))
else
    println("Box entry aborted.")
)
)
procedure( boxForm( form map )
    if( map then
        eval(form)->layer->value = "y2"
        eval(form)->purpose->value = "drawing"
        println("map form")
    else
        println("unmap form")
    )
)
hiCreateOptionsForm(
    'myForm
    "Layer Entry"
    list(hiCreateStringField(
        ?name 'layer
        ?prompt "purpose"
    )
        hiCreateStringField(
            ?name 'purpose
            ?prompt "purpose"
        )
    )
    nil
)
procedure( test( )
    enterBox(
        ?prompts list("Enter the first corner of your box."
            "Enter the last corner of your box.")
        ?doneProc "boxDone"
        ?formProc "boxForm"
        ?alwaysMap t
        ?form eval(myForm)
    )
)
hiSetBindKey("Layout" "<Key>5" "test( )")
```

The prompt “Enter the first corner of your box” is displayed on the CIW prompt line. `myForm` is a form containing two string items, one for the layer and one for the purpose. The form is immediately displayed, regardless of user ability mode. The layer-purpose pair is initially `y2` and `drawing`. You change the layer to `ndiff` and the purpose to `pin`.

When you enter point *p1* the prompt changes to “Enter the last corner of your box.” A rubberband box appears from *p1* to the cursor. Enter point *p2* and `doneProc` prints out the box points:

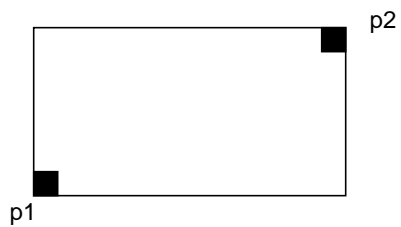
```
Box entered was ( p1 p2 ). Layer is ("ndiff" "pin").
```

The `enterfunction` returns the box points:

Cadence User Interface SKILL Functions Reference

User Entry Functions

(p1 p2)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterCircle

```
enterCircle(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_bBox / nil
```

Description

Enters a circle in the current window.

Arguments

<i>l_promptList</i>	List of prompt strings to appear on the CIW prompt line. If supplied, only the first two strings are used. The first string corresponds to the center, which is the first point entered, and the second string corresponds to a point on the circumference.
<i>l_pointList</i>	List of no more than two points. If both points are supplied, they are returned after adjustment with no further interaction. If one point is supplied, it is adjusted and you are prompted to enter the second point. The first point entered is the center of the circle. After entry, a rubberbanding circle appears about the center and passing through the cursor. When the second point is entered, the bounding box of the circle is computed and returned.
<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options that are relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Value Returned

<i>l_bBox</i>	Returns a list containing the bounding box of the circle.
<i>nil</i>	Returns <code>nil</code> if the circle is not created.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( circleInit( w )
  leSetEntryLayer( list("pdiff" "drawing"))
)
```

Cadence User Interface SKILL Functions Reference

User Entry Functions

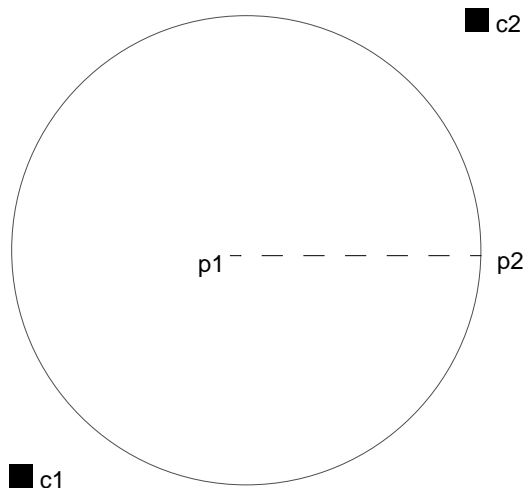
```
procedure( circleDone( w done pts )
  if( done then
    printf("Circle entered was %L. Layer is %L.\n" pts
      leGetEntryLayer( ))
  else
    println("Circle entry aborted.")
  )
)
enterCircle( ?prompts list( "Enter the circle center."
  "Enter a point on the circumference." )
  ?doneProc "circleDone" ?initProc "circleInit"
)
```

The prompt “Enter the circle center” is displayed on the CIW prompt line. The `initProc` forces the window’s current layer and purpose pair to be (“pdiff” “drawing”). When you enter point *p1* the prompt changes to “Enter a point on the circumference.” A rubberband circle appears with center *p1* and passing through the cursor. Enter point *p2* and `doneProc` prints the circle points:

Circle entered was (c1 c2). Layer is ("pdiff" "drawing").

Note that (c1 c2) is the bounding box of the circle entered. The `enterfunction` returns a list of the circle points:

(c1 c2)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterDonut

```
enterDonut(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_point_list / nil
```

Description

Enter a donut in the current window.

Arguments

<i>l_promptList</i>	List of prompt strings to appear on the CIW prompt line. If supplied, only the first three strings are used. The first string corresponds to the center, which is the first point entered. The second string corresponds to a point on the circumference of the donut hole. The third string corresponds to a point on the outer circumference of the donut.
<i>l_pointList</i>	<p>List of no more than three points. If all points are specified, they are returned after adjustment. If not, the supplied points are adjusted and you are prompted for the remaining points.</p> <p>The first point is the center of the donut. After the first point is entered, a rubberbanding circle is drawn with that center point and passing through the cursor. When the second point is entered, the circle becomes permanent, and another rubberbanding circle is drawn. After the third point is entered, the three points are returned in a list.</p>

Cadence User Interface SKILL Functions Reference

User Entry Functions

The second and third points determine the hole radius and the outer radius of the donut and can be specified in either order. The returned list orders the points as center, inner radius, and outer radius.

<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with hiGetCurrentCmd .

Cadence User Interface SKILL Functions Reference

User Entry Functions

Value Returned

l_point_list Returns the entered points as a list containing the coordinates for the donut.

nil Returns *nil* if the circle is not drawn.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( donutAddPt( w pts )
  prog( (p)
    p = car(pts)
    if( car(p) < 0 || cadr(p) < 0 then
      warn("Donut center is not in first quadrant.\n") )
  )
)

procedure( donutDone( w done pts )
  if( done then
    printf("Donut entered was %L.\n" pts)
  else
    println("Donut entry aborted.")
  )
)

enterDonut( ?prompts list( "Enter the donut center."
  "Enter a point on the circumference of the hole."
  "Enter a point on the circumference of the donut.")
  ?doneProc "donutDone" ?addPointProc "donutAddPt"
)
```

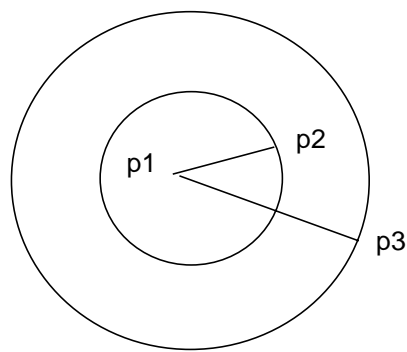
The prompt “Enter the donut center” is displayed on the CIW prompt line. When you enter point *p1* the prompt changes to “Enter a point on the circumference.” If *p1* is not in the first quadrant, the message “Donut center is not in the first quadrant.” appears. A rubber band circle appears with center *p1* and passing through the cursor. Enter point *p2*. The circle becomes permanent, and a new rubberbanding circle appears through the cursor. Enter point *p3*. *doneProc* prints the donut points:

Donut entered was (*p1 p2 p3*).

The enterfunction returns a list of the donut points:

Cadence User Interface SKILL Functions Reference
User Entry Functions

(p1 p2 p3)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterEllipse

```
enterEllipse(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_bBox
```

Description

Enters an ellipse in the current window.

An ellipse is defined by its bounding box, therefore, only horizontally and vertically oriented ellipses can be entered.

Arguments

<i>l_promptList</i>	List of prompt strings to appear on the CIW prompt line. If supplied, only the first two strings are used. Each string corresponds to one of the two points needed to specify the bounding box of the ellipse.
<i>l_pointList</i>	List of no more than two points specifying the ellipse bounding box. If both points are specified, the box is returned after adjustment. If not, you are prompted for the remaining point or points. If one point is supplied, it is adjusted as if you had entered it. After you enter the first point, a rubberbanding ellipse is drawn inside the bounding box defined by the first point and the cursor's current position. After the second point is entered the function returns a list of two points. The list returned is normalized so that the first point corresponds to the lower left corner of the box; the second is the upper right corner.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options that are relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with hiGetCurrentCmd .

Value Returned

<i>l_bBox</i>	Returns the two digitized points as coordinates in user units. These points represent the bounding box containing the ellipse. It is normalized so that the first point corresponds to the lower left corner of the bounding box and the second is the upper right corner.
---------------	--

Cadence User Interface SKILL Functions Reference

User Entry Functions

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( ellipseDone( w done pts )
  if( done then
    if( stringp(pts) then
      pts = list( pts "drawing" )
      leGetEntryLayer(pts)
      printf("Current layer changed to %L.\n" pts)
    else
      printf("Ellipse entered was %L. Layer is %L.\n" pts
        leGetEntryLayer( ))
    else
      println("Ellipse entry aborted.")
  )
)
enterEllipse( ?prompts
  list( "Enter the first ellipse point."
    "Enter the second ellipse point." )
  ?doneProc "circleDone" ?acceptString t
)
```

The prompt “Enter the first ellipse point” is displayed on the CIW prompt line. When you enter point *p1* the prompt changes to “Enter the second ellipse point”. A rubberband ellipse appears with a bounding box constructed from *p1* and the current cursor location. Type “thinor” into the CIW. *doneProc* notes the change:

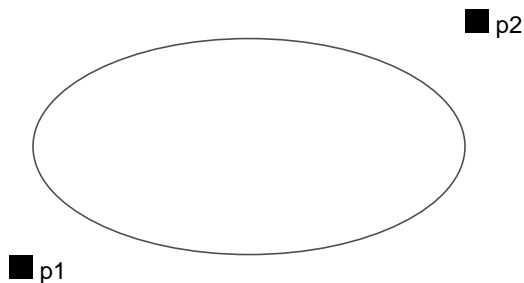
Current layer changed to ("thinor" "drawing").

Note that the enterfunction is still active, as shown by the rubberband ellipse and the CIW prompt. Enter point *p2*. *doneProc* prints the ellipse points:

Ellipse entered was (p1 p2). Layer is ("thinor" "drawing").

Note that (*p1 p2*) is the bounding box of the circle entered. The enterfunction returns the ellipse points:

(p1 p2)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterLine

```
enterLine(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?wantPoints x_pointLimit]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_point_list / nil
```

Description

Enters a multisegment line in the current window.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed in the CIW.
<i>l_pointList</i>	<p>List of points to start the line. If specified, all points are adjusted. If the list terminates with two identical points, the adjusted points are returned with no further interaction. Otherwise, the given points are displayed with connecting lines and a rubberbanding line is drawn from the last point to the current cursor position. Each successive point entered causes a new permanent segment between that point and the previous point.</p> <p>The function terminates when two identical points are entered or when the function is explicitly terminated with <u>finishEnterFun</u> or <u>cancelEnterFun</u>.</p>
<i>x_pointLimit</i>	Number specifying the maximum number of input points to be used. The function returns the list of points after it reaches this number. In this case, the last two points do not need to be identical to terminate the function.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>s_form</i>	Options form created with <u>hiCreateOptionsForm</u> . This form is displayed when you call <u>hiToggleEnterForm</u> . It contains options that are relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubberband lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Value Returned

<i>l_point_list</i>	Returns the points you entered as a list containing the coordinates for a line.
<i>nil</i>	Returns <code>nil</code> if the line is not entered.

Cadence User Interface SKILL Functions Reference

User Entry Functions

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( lineDone( w done pts )
  if( done then
    printf("Line entered was %L.\n" pts)
  else
    println("Line entry aborted.")
  )
)
enterLine( ?prompts
  list( "Enter the first point."
        "Enter the next point." )
  ?doneProc "lineDone"
)
```

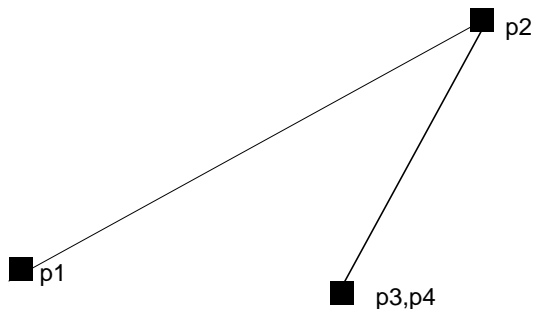
The prompt “Enter the first point” is displayed on the prompt line. When you enter point *p1*, the prompt changes to “Enter the next point”. A rubber band line appears between *p1* and the current cursor location.

Enter point *p2* and the first segment becomes permanent, while the rubberbanding line now attaches the cursor to *p2*. The prompt in the CIW reads the same, “Enter the next point”, because the second prompt in the list is used for any points after the first. Continue with *p3*. Enter *p4* at the same location as *p3*. *doneProc* prints the point list:

Line entered was (p1 p2 p3 p4).

The enterfunction returns the point list:

(p1 p2 p3 p4)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterNumber

```
enterNumber(  
    [?prompts l_promptList]  
    [?form s_form]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?alwaysMap g_alwaysMap]  
    [?cmdName t_cmdName]  
)  
=> f_number
```

Description

Prompts you to enter a number.

When you type a carriage return, any characters accumulated in the input line are returned as a number in floating-point format.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed in the CIW. Only the first prompt in the list is used.
<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm and contains options that are relevant to the enterfunction and its caller. The form can be displayed any time before you enter a carriage return to terminate the enterfunction.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated. The number is passed as the third argument to the done proc.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.

Cadence User Interface SKILL Functions Reference

User Entry Functions

t_cmdName Name you associate with the enterfunction. If the command name is set, it can be accessed with hiGetCurrentCmd.

Value Returned

f_number Returns the number you typed in floating-point format.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
enterNumber( ?prompts
  list( "Enter any number." )
)
```

The prompt "Enter any number" is displayed on the CIW prompt line. When you type a number followed by a carriage return, the enterfunction returns that number.

Cadence User Interface SKILL Functions Reference

User Entry Functions

enterPath

```
enterPath(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?wantPoints x_pointLimit]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?pathStyle t_pathStyle]  
    [?pathWidth f_pathWidth]  
    [?beginExtent f_beginExtent]  
    [?endExtent f_endExtent]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_point_list / nil
```

Description

Enters a path in the current window.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed.
<i>l_pointList</i>	List of points to start the path. If specified, all points are adjusted. If the list terminates with two identical points, the adjusted points are returned with no further interaction. If not, the path is displayed using the given points, and a rubberbanding path segment is drawn from the last point to the current cursor position. Each successive point entered causes a new permanent segment between that point and the previous point. The function terminates when two identical points are entered, or when the function is explicitly terminated with <u>finishEnterFun</u> or <u>cancelEnterFun</u> .

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>x_pointLimit</i>	Number specifying the maximum number of points to be used. If the <i>x_pointLimit</i> argument is supplied, the function returns the list of points after it reaches this number. In this case, the last two points do not need to be identical to terminate the function.								
<i>s_form</i>	Options form created with <i>hiCreateOptionsForm</i> . This form is displayed when you call <i>hiToggleEnterForm</i> . It contains options that are relevant to the enterfunction and its caller.								
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.								
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.								
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.								
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.								
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.								
<i>t_pathStyle</i>	String defining the path style. It can take one of the following values: <table> <tr> <td>T or Truncate</td><td>The path is square ended with no extension.</td></tr> <tr> <td>E or Extended</td><td>The path is square ended with an extension half of the width on each end.</td></tr> <tr> <td>R or Rounded</td><td>The path is round ended with no extension.</td></tr> <tr> <td>V or VariableEnd</td><td>The path is square ended with its extension specified by the <i>f_beginExtent</i> and <i>f_endExtent</i> arguments.</td></tr> </table>	T or Truncate	The path is square ended with no extension.	E or Extended	The path is square ended with an extension half of the width on each end.	R or Rounded	The path is round ended with no extension.	V or VariableEnd	The path is square ended with its extension specified by the <i>f_beginExtent</i> and <i>f_endExtent</i> arguments.
T or Truncate	The path is square ended with no extension.								
E or Extended	The path is square ended with an extension half of the width on each end.								
R or Rounded	The path is round ended with no extension.								
V or VariableEnd	The path is square ended with its extension specified by the <i>f_beginExtent</i> and <i>f_endExtent</i> arguments.								
<i>f_pathWidth</i>	Width of the path in user units.								
<i>f_beginExtent</i>	Distance from the first point that the path extends to.								
<i>f_endExtent</i>	Distance from the last point that the path extends to.								

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Value Returned

<i>l_point_list</i>	Returns the points you entered as a list containing the coordinates for a path.
<i>nil</i>	Returns <code>nil</code> if the path is not created.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( pathDone( w done pts )
  if( done then
    printf("Path entered was %L.\n" pts)
  else
    println("Path entry aborted.")
  )
)

enterPath( ?prompts
  list( "Enter the first point."
    "Enter the next point." )
  ?doneProc "pathDone"
  ?wantPoints 4
  ?pathWidth 6. ?pathStyle "Truncate"
)
```

Cadence User Interface SKILL Functions Reference

User Entry Functions

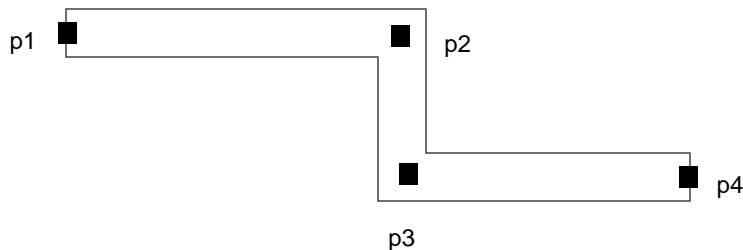
The prompt “Enter the first point” is displayed on the CIW prompt line. When you enter point *p1* the prompt changes to “Enter the next point.” A rubber band path with width 6 appears between *p1* and the current cursor location.

Enter point *p2* and the first path segment becomes permanent, while the rubberbanding path segment now attaches the cursor to *p2*. The prompt remains the same, “Enter the next point,” because the second prompt in the list is used for any points after the first. Continue with *p3* and *p4*. `doneProc` prints out the list of points:

```
Path entered was ( p1 p2 p3 p4 ).
```

The `enterfunction` returns the list of points:

```
( p1 p2 p3 p4 )
```



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterPoint

```
enterPoint(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_coord_pair
```

Description

Prompts you to enter a single point.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed. The first string in the list is displayed until you enter a point. Any other strings are ignored.
<i>l_pointList</i>	List with no more than one point in it. If supplied, the point is adjusted and returned with no further interaction. If not supplied, the first string in <i>l_promptList</i> is displayed on the CIW prompt line asking you to enter the point.
<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options that are relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted. Note that since entering the first point terminates the function, this procedure is effectively never called.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Value Returned

l_coord_pair Returns the adjusted point that you entered.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( pointDone( w done pts )
  if( done then
    printf("Point entered was %L.\n" car(pts))
  else
    println("Point entry aborted.")
  )
)
enterPoint( ?prompts
  list( "Enter a point.")
  ?doneProc "pointDone"
)
```

The prompt “Enter a point” is displayed on the prompt line. Type the following into the CIW:

2:6

The enterfunction terminates, and *doneProc* prints out the point:

Cadence User Interface SKILL Functions Reference

User Entry Functions

Point entered was (2.000 6.000).

The enterfunction returns the point:

(2.000 6.000)

Cadence User Interface SKILL Functions Reference

User Entry Functions

enterPoints

```
enterPoints(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?wantPoints x_pointLimit]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_coord_pair
```

Description

Prompts you to enter a list of points.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed for successive points.
<i>l_pointList</i>	List of points to be preloaded into the enterfunction. If this list is terminated by two identical points, the function returns the adjusted points with no further interaction. If not, any points in this list are added. The points are not displayed, and there is no rubber band line connecting any of the points. The function is terminated in the same manner as other multi-point enterfunctions.
<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options that are relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>x_pointLimit</i>	Maximum number of points to be accepted by this function. Collects points until the number of points specified by <i>x_pointLimit</i> is reached or until you finish or cancel the enterfunction. Nothing is drawn by the enterfunction.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Value Returned

l_coord_pair Returns the adjusted points entered as a list of points.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( pointsDone( w done pts )
  if( done then
    printf("Points entered were %L.\n" pts)
  else
    println("Point entry aborted.")
  )
)
```

Cadence User Interface SKILL Functions Reference

User Entry Functions

```
enterPoints( ?prompts
  list( "Enter the first point."
    "Enter the next point." )
  ?doneProc "pointsDone"
)
```

The prompt “Enter the first point” is displayed on the prompt line. When you enter point *p1* the prompt changes to “Enter the next point.” Unlike `enterLine`, no rubber band line appears between *p1* and the current cursor location.

Enter point *p2*. The prompt reads the same— “Enter the next point”— because the second prompt in the list is used for any points after the first. Continue with *p3*.

Type in the CIW:

```
finishEnterFun( )
```

`doneProc` prints the points entered:

```
Points entered were ( p1 p2 p3 ).
```

The `enterfunction` returns the point list:

```
( p1 p2 p3 )
```

Cadence User Interface SKILL Functions Reference

User Entry Functions

enterPolygon

```
enterPolygon(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?wantPoints x_pointLimit]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_point_list / nil
```

Description

Digitizes a polygon in the current window.

The function does not allow entry of points that result in a self-intersecting polygon. It also removes any colinear points.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed.
<i>l_pointList</i>	List of points to start the polygon. If specified, all points are adjusted. If the list terminates with a point identical to the first point, the adjusted points are returned with no further interaction. If not, the given points are displayed with connecting lines, and a rubberbanding line is drawn from the last point to the current cursor position. Each successive point entered causes a new permanent segment between that point and the previous point. The function terminates when a point identical to the first point is entered or when the function is explicitly terminated with <u>finishEnterFun</u> or <u>cancelEnterFun</u> .
<i>x_pointLimit</i>	Maximum number of points to be specified. If the <i>x_pointLimit</i> argument is supplied, the function returns the

Cadence User Interface SKILL Functions Reference

User Entry Functions

list of points after it reaches this number. In this case, the last two points do not need to be identical to terminate the function.

s_form

Options form created with [hiCreateOptionsForm](#). This form is displayed when you call [hiToggleEnterForm](#). It contains options that are relevant to the enterfunction and its caller.

t_addProcName

Name of the SKILL function to be called when a point is entered.

t_delProcName

Name of the SKILL function to be called when a point is deleted.

t_initProcName

Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.

t_doneProcName

Name of the SKILL function to be called when the enterfunction is terminated.

t_formProcName

Name of the SKILL function to be called when the options form is displayed.

g_dontDraw

Either `t` or `nil` to determine whether or not to draw rubber band lines.

g_alwaysMap

Either `t` or `nil` to determine whether or not to immediately display the options form.

g_acceptString

Either `t` or `nil` to determine whether or not to accept string input without terminating the enterfunction.

g_acceptNumber

Either `t` or `nil` to determine whether or not to accept numeric input without terminating the enterfunction.

g_noInfix

Either `t` or `nil` to determine whether or not to disable infix mode.

t_cmdName

Name you associate with the enterfunction. If the command name is set, it can be accessed with [hiGetCurrentCmd](#).

Value Returned

l_point_list

Returns the list of points that have been entered. Note that if the function is terminated by returning to the initial point, that point appears at the beginning *and* at the end of the returned list.

Cadence User Interface SKILL Functions Reference

User Entry Functions

`nil` Returns `nil` If the function is canceled.

Note: Retrieve data through the callback procedure `t_doneProcName` instead of using the return value for this enterfunction.

Example

```
procedure( polyDone( w done pts )
  if( done then
    printf("Polygon entered was %L.\n" pts)
    dbCreatePolygon( w->editCellView
      list( "cut" "drawing" )
      pts )
  else
    println("Polygon entry aborted.")
  )
)
enterPolygon( ?prompts
  list( "Enter the first point."
    "Enter the next point." )
  ?doneProc "polyDone"
  ?dontDraw t
)
```

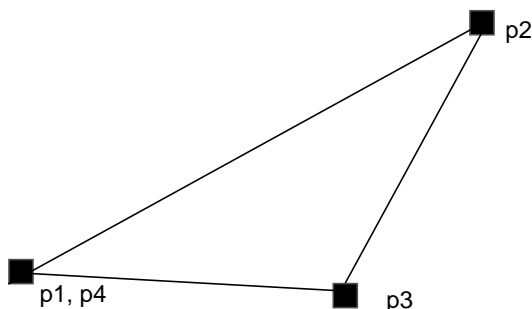
The prompt “Enter the first point” is displayed on the prompt line. When you enter point *p1* the prompt changes to “Enter the next point.” No rubber band line appears between *p1* and the current cursor location because the `dontDraw` flag is specified.

Enter point *p2*. The prompt reads the same—“Enter the next point”—because the second prompt in the list is used for any points after the first. Continue with *p3*. Now enter *p4* at the same location as *p1*. `doneProc` creates the polygon on the `cut` drawing layer and prints out the points entered. Note that *p4* is the same point as *p1*.

The polygon entered was (*p1 p2 p3 p4*).

The enterfunction returns the point list:

(*p1 p2 p3 p4*)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterScreenBox

```
enterScreenBox(  
    )  
=> l_bBox
```

Description

Lets you enter a box in screen coordinates.

Most enterfunctions return coordinates in user units, so this function provides a way to specify screen coordinates when necessary. When `enterScreenBox` is invoked, it displays a small box on the root window (denoting that it is in `ScreenBox` mode). Press the mouse button at the first point and hold down. While the button is down a rubberbanding box is displayed. When you release the mouse button, the box is returned. Note that this is different from the behavior of `enterBox`, which uses two clicks, one for each corner.

`enterScreenBox` is not nestable and suspends all other enterfunctions until it completes.

Arguments

None.

Value Returned

`l_bBox` Returns the bounding box in screen coordinates that you entered.

Note: Retrieve data through the callback procedure `t_doneProcName` instead of using the return value for this enterfunction.

Example

```
enterScreenBox( )
```

The cursor changes to a small box.

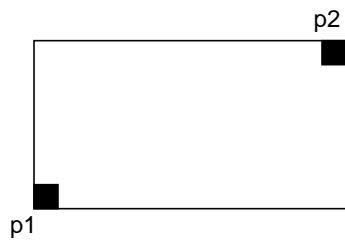
Press the mouse button at `p1`. While the button is down, a rubberbanding box connects `p1` to the current cursor location. Release the mouse button at the desired location.

The function terminates and returns the selected points in screen coordinates:

Cadence User Interface SKILL Functions Reference

User Entry Functions

(p1 p2)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterSegment

```
enterSegment(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_point_list / nil
```

Description

Lets you enter a segment into the current window.

The function accepts two points. After the first point is digitized, a rubberbanding line appears between the first point and the cursor's current location.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed. Only the first two strings are used.
<i>l_pointList</i>	List of no more than two points. If both points are supplied, they are adjusted and returned with no further interaction. If one point is supplied, it is adjusted and displayed as if the user had entered it. After the first point is entered a rubberbanding line appears between the point and the current cursor location. When the second point is entered, the function terminates and returns the list of points entered. The segment obeys the current snap mode and snap shape rule when in Graphics Editor windows.
<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options that are relevant to the enterfunction and its caller.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Value Returned

<i>l_point_list</i>	Returns a list containing the coordinates for a segment.
<i>nil</i>	Returns <code>nil</code> if the function is cancelled.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Cadence User Interface SKILL Functions Reference

User Entry Functions

Example

```
procedure( segDone( w done pts )
  if( done then
    printf("Segment entered was %L.\n" pts)
  else
    println("Segment entry aborted.")
  )
)
procedure( myEnterSegment( )
  enterSegment( ?prompts
    list( "Enter the first point."
      "Enter the second point." )
    ?doneProc "segDone"
  )
)
window(1)->infixOn = t
hiSetBindKey("Schematics" "<Key>s" "myEnterSegment( )")
```

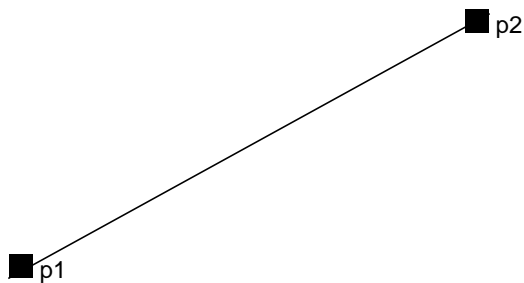
The last two statements of the procedure set infix mode on and bind the *s* key to the procedure `myEnterSegment`. Type the *s* key in a Schematics window at location *p1*. Because of infix mode, *p1* is accepted as the first point for the `enterSegment` call. The rubberbanding line immediately appears between *p1* and the cursor location. The prompt "Enter the second point" is displayed on the prompt line. The first prompt is skipped since the first point is already entered.

Enter point *p2*. `doneProc` prints out the points entered:

```
Segment entered was ( p1 p2 ).
```

The `enter` function returns the list of points:

```
( p1 p2 )
```



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterMultiRep

```
enterMultiRep(  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> l_point_list / nil
```

Description

Identical to `enterSegment` except that the two points can be in windows viewing different cellviews.

This is the only enterfunction that allows this capability. All others accept input only in windows viewing the same Cellview (known as *equivalent* windows).

Arguments

<code>l_promptList</code>	List of prompt strings to be displayed in the CIW. Only the first two strings are used.
<code>l_pointList</code>	<p>List of no more than two points. If both points are supplied, they are adjusted and returned with no further interaction. Note that in this case, the two points are both assumed to be in the current window. If one point is supplied, it is adjusted and displayed as if the user had entered it. After the first point is entered a rubberbanding line appears between the point and the current cursor location.</p> <p>The rubberbanding line appears only while the cursor is in windows equivalent to the window in which the first point was entered. When the second point is entered, the function terminates and returns the list of points entered. If the second</p>

Cadence User Interface SKILL Functions Reference

User Entry Functions

point is entered in a window that is not equivalent to the first, it is returned in coordinates corresponding to the window in which it was entered. Normally this is not a problem. If the two windows, however, have different database-to-user-unit conversion factors, the result might not be as expected. The segment obeys the current snap mode and snap shape rule when in Graphics Editor windows.

<i>s_form</i>	Options form created with <u>hiCreateOptionsForm</u> . This form is displayed when you call <u>hiToggleEnterForm</u> . It contains options that are relevant to the enterfunction and its caller.
<i>t_addProcName</i>	Name of the SKILL function to be called when a point is entered.
<i>t_delProcName</i>	Name of the SKILL function to be called when a point is deleted.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_dontDraw</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>g_acceptString</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.
<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with <u>hiGetCurrentCmd</u> .

Cadence User Interface SKILL Functions Reference

User Entry Functions

Value Returned

l_point_list Returns a list of two points.

nil Returns *nil* if the function is cancelled.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
procedure( segDone( w done pts )
  if( done then
    printf("Segment extends from %L to %L.\n"
           w hiGetCurrentWindow())
  else
    println("Segment entry aborted.")
  )
)

procedure( segInit( )
  w = hiGetCurrentWindow( )
  firstWindow =
    evalstring(cadr(parseString(sprintf(nil "%L" w) ":")))
)

procedure( myEnterSegment( )
  segInit( )
  enterMultiRep(
    ?prompts list( "Enter the first point."
                  "Enter the second point." )
    ?doneProc "segDone"
    ?noInfix t
  )
)

window(1)->infixOn = t
hiSetBindKey("Schematics" "<Key>s" "myEnterSegment( )")
```

These last two statements set the infix mode on and bind the *s* key to the procedure *myEnterSegment*. Type the *s* key in a Schematics window at location *p1*. Because of the *noInfix* flag, the *infixOn* flag is ignored. The prompt "Enter the first point" is displayed in the CIW. Enter point *p1* in *window(2)*. The rubberbanding line appears between *p1* and the cursor location. The prompt "Enter the second point" is displayed on the CIW prompt line.

Enter point *p2* in *window(3)*, which views a different cellview. *doneProc* prints out the specified message:

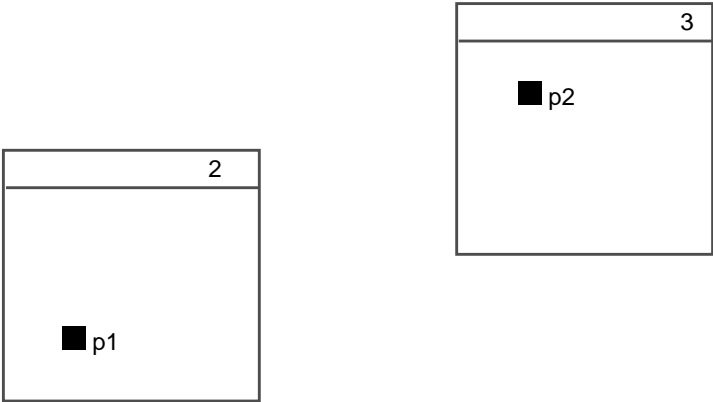
```
Segment extends from window:2 to window:3.
```

The enterfunction returns the point list:

Cadence User Interface SKILL Functions Reference

User Entry Functions

(p1 p2)



Cadence User Interface SKILL Functions Reference

User Entry Functions

enterString

```
enterString(  
    [?prompts l_promptList]  
    [?form s_form]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?alwaysMap g_alwaysMap]  
    [?cmdName t_cmdName]  
)  
=> t_string
```

Description

Prompts you to enter a string.

The function terminates when you press a carriage return in the CIW.

Arguments

<i>l_promptList</i>	List of prompt strings to be displayed in the CIW. Only the first string is used.
<i>s_form</i>	Options form created with hiCreateOptionsForm . This form is displayed when you call hiToggleEnterForm . It contains options that are relevant to the enterfunction and its caller.
<i>t_initProcName</i>	Name of the SKILL function to be called immediately after the enterfunction is started and before any data entry.
<i>t_doneProcName</i>	Name of the SKILL function to be called when the enterfunction is terminated.
<i>t_formProcName</i>	Name of the SKILL function to be called when the options form is displayed.
<i>g_alwaysMap</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<i>t_cmdName</i>	Name you associate with the enterfunction. If the command name is set, it can be accessed with hiGetCurrentCmd .

Cadence User Interface SKILL Functions Reference

User Entry Functions

Value Returned

t_string Returns the string you typed.

Note: Retrieve data through the callback procedure *t_doneProcName* instead of using the return value for this enterfunction.

Example

```
enterString( ?prompts list( "Enter layer name." ))
```

The prompt “Enter layer name” is displayed on the prompt line. When you type a string followed by a carriage return, the enterfunction returns that string.

Cadence User Interface SKILL Functions Reference

User Entry Functions

addPoint

```
addPoint(  
    l_point  
)  
=> t / nil
```

Description

Adds the point to the enterfunction in the current window.

`addPoint` is often used in conjunction with `hiGetCommandPoint` to add the point where you last clicked to an enterfunction.

Arguments

<code>l_point</code>	Point in user units to add to the current enterfunction.
----------------------	--

Value Returned

<code>t</code>	Returns <code>t</code> if the point is added
<code>nil</code>	Returns <code>nil</code> if the point is not added.

Example

```
procedure( pointDone( w done pts )  
    if( done then  
        printf("Point entered was %L.\n" car(pts))  
    else  
        println("Point entry aborted.")  
    )  
)  
enterPoint( ?prompts  
    list( "Enter a point.")  
    ?doneProc "pointDone"  
)
```

The prompt "Enter a point." is displayed on the CIW prompt line and the point is entered:

```
addPoint(2:6)
```

The enterfunction terminates and `doneProc` prints the specified message:

```
Point entered was (2.000 6.000).
```

The enterfunction returns the point list:

Cadence User Interface SKILL Functions Reference
User Entry Functions

(2.000 6.000)

Cadence User Interface SKILL Functions Reference

User Entry Functions

preXY

```
preXY(  
    l_point  
)  
=> t
```

Description

Causes an `addPoint(l_point)` command to be added to the command queue in order to be logged and executed at the next toplevel to simulate the user selecting a point.

Arguments

<i>l_point</i>	Point to be added.
----------------	--------------------

Value Returned

t	Returns t after the point command is added to the queue.
---	--

Example

This example enters a polygon of five points where the first three points are already calculated but the `addPointProc` procedure is called for all the points, even the first three.

```
MyPointList = { preXY( 0.0:0.0 ) preXY( 5.0:0.0 ) preXY( 5.0:5.0 )  
    enterPolygon( ?prompts list("enter point")  
        ?wantPoints 5  
        ?addPointProc "MyAddPointProc"  
    )  
}
```

Reference

[addPoint](#)

Cadence User Interface SKILL Functions Reference

User Entry Functions

deletePoint

```
deletePoint(  
    )  
=> t / nil
```

Description

Deletes a point from the enterfunction in the current window.

Arguments

None.

Value Returned

t	Returns t if the point was successfully deleted from the pending enterfunction.
nil	Returns nil if the current window is not in an enterfunction.

Example

```
procedure( segDone( w done pts )  
    if( done then  
        printf("Segment entered was %L.\n" pts)  
    else  
        println("Segment entry aborted.")  
    )  
)  
  
procedure( myEnterSegment( )  
    enterSegment( ?prompts  
        list( "Enter the first point."  
            "Enter the second point." )  
        ?doneProc "segDone"  
    )  
)  
window(1)->infixOn = t  
hiSetBindKey("Schematics" "<Key>s" "myEnterSegment( )")
```

The last two statements set the infix mode on and bind the s key to the procedure myEnterSegment. Type the s key in a Schematics window at location *p0*. Because of infix mode, *p0* is taken as the first point for the enterSegment call. The rubberbanding line immediately appears between *p1* and the cursor location. The prompt "Enter the second point" is displayed on the CIW prompt line. The first prompt is skipped since the first point is already entered. Type

Cadence User Interface SKILL Functions Reference

User Entry Functions

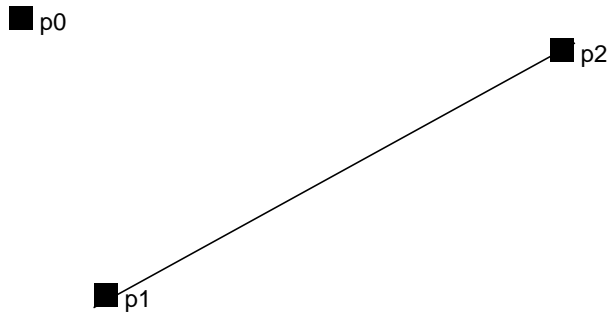
`deletePoint()`

The rubberbanding line goes away, and the CIW prompt changes to “Enter the first point.” Enter point *p1*. The rubberbanding line reappears from *p1* to the cursor, and the prompt changes to “Enter the second point.” Enter *p2*. `doneProc` prints out the specified message:

Segment entered was (*p1 p2*).

The enterfunction returns the point list:

(*p1 p2*)



Cadence User Interface SKILL Functions Reference

User Entry Functions

cancelEnterFun

```
cancelEnterFun(  
    [g_form]  
)  
=> t / nil
```

Description

Cancels the enterfunction in the current window.

Arguments

<i>g_form</i>	Form handle used to locate the enterfunction to cancel. Normally the enterfunction that is cancelled is the one active in the current window. If you want to cancel a specific enterfunction, whether or not it is in the current window, there is no way to tell which window is the correct one. So you pass the same form that was originally supplied to the enterfunction to <code>cancelEnterFun</code> , and it cancels that specific enterfunction regardless of whether it is in the current window or not.
---------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the enterfunction is cancelled.
<i>nil</i>	Returns <i>nil</i> if there is no active enterfunction in the current window.

Example

```
enterSegment(?prompts list("First" "Second"))
```

The prompt “First” is displayed in the CIW. Type

```
cancelEnterFun( )
```

The enterfunction is terminated, the prompt goes away, and *nil* is returned.

Cadence User Interface SKILL Functions Reference

User Entry Functions

finishEnterFun

```
finishEnterFun(  
    [g_form]  
)  
=> t / nil
```

Description

Completes an enterfunction if possible.

Some commands cannot be finished until you enter all the required points. When `finishEnterFun` is called in such a situation, the enterfunction is cancelled even though `finishEnterFun` returns `t`.

Arguments

<i>g_form</i>	Form handle used to locate the enterfunction to finish. Normally the enterfunction finished is the one active in the current window. If you want to finish a specific enterfunction, whether or not it is the current window, there is no way to tell which window is the correct one. So you pass the same form that was originally supplied to the enterfunction to <code>finishEnterFun</code> , and it finishes that specific enterfunction regardless of whether it is in the current window or not.
---------------	---

Value Returned

<code>t</code>	Returns <code>t</code> if <code>finishEnterFun</code> terminates the enterfunction in the current window.
<code>nil</code>	Returns <code>nil</code> if it could not terminate the enterfunction.

Example

```
enterPoints(?prompts list("First" "Next"))
```

The prompt “First” is displayed in the CIW. Enter points *p1*, *p2*, and *p3*. Now type

```
finishEnterFun( )
```

The enterfunction is terminated, the prompt goes away, and (*p1 p2 p3*) is returned.

Cadence User Interface SKILL Functions Reference

User Entry Functions

applyEnterFun

```
applyEnterFun(  
    [g_form]  
)  
=> t / nil
```

Description

Calls `doneProc`, passes it the points you entered, resets the enterfunction, and calls `initProc`.

This function does not return from the active enterfunction.

`applyEnterFun` can be used in conjunction with `doneProc` to allow you to enter many shapes without having to start another enterfunction.

Arguments

<i>g_form</i>	Form handle used to locate the enterfunction to apply. Normally the enterfunction applied is the one active in the current window. If you want to apply a specific enterfunction, whether or not it is the current window, there is no way to tell which window is the correct one. So you pass the same form that was originally supplied to the enterfunction to <code>applyEnterFun</code> , and it applies that specific enterfunction regardless of whether it is in the current window or not.
---------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the enterfunction is begun.
<i>nil</i>	Returns <i>nil</i> if the enterfunction is not begun.

Example

```
procedure(donePoints( w done pts)  
    if( done then  
        printf("Points = %L\n" pts)  
    else  
        println("Point entry aborted.")  
    )  
)  
enterPoints(  
    ?prompts list("First" "Next") ?doneProc "donePoints")
```

Cadence User Interface SKILL Functions Reference

User Entry Functions

The prompt "First" is displayed in the CIW. Enter point $p1$. The prompt changes to "Next. " Enter $p2$ and $p3$. Now type

```
applyEnterFun( )
```

The enterfunction is not terminated, the prompt changes back to "First, " and "Points = ($p1$ $p2$ $p3$)" is printed out. Now enter points $p4$, $p5$, and $p6$, followed by

```
finishEnterFun( )
```

The enterfunction is terminated, the prompt goes away, the message "Points = ($p4$ $p5$ $p6$)" is printed out, and ($p4$ $p5$ $p6$) is returned.

Cadence User Interface SKILL Functions Reference

User Entry Functions

changeEnterFun

```
changeEnterFun(  
    s_enterFun  
    [?prompts l_promptList]  
    [?points l_pointList]  
    [?wantPoints x_pointLimit]  
    [?form s_form]  
    [?addPointProc t_addProcName]  
    [?delPointProc t_delProcName]  
    [?initProc t_initProcName]  
    [?doneProc t_doneProcName]  
    [?formProc t_formProcName]  
    [?pathStyle t_pathStyle]  
    [?pathWidth f_pathWidth]  
    [?beginExtent f_beginExtent]  
    [?endExtent f_endExtent]  
    [?dontDraw g_dontDraw]  
    [?alwaysMap g_alwaysMap]  
    [?acceptString g_acceptString]  
    [?acceptNumber g_acceptNumber]  
    [?noInfix g_noInfix]  
    [?cmdName t_cmdName]  
)  
=> t / nil
```

Description

Changes the current enterfunction to another one.

Note: Since this function uses the current window to determine which enterfunction to change, make sure to properly set the current window before using this function. See [hiSetCurrentWindow\(\)](#) for information about setting the current window.

You can use `changeEnterFun` to get immediate response to any changes the user makes to the values on the option forms. For more information, see [Option Forms](#).

`s_enterFun` is the symbol of the enterfunction you want to change. For example, if you want to change to an *enterBox*, you specify the argument as 'enterBox.

`changeEnterFun` should always be provided with a `?doneProc` argument.

`changeEnterFun` cancels the current enterfunction and then changes it to the new one. The call to the original enterfunction remains blocked until the changed enterfunction exits. When the new enterfunction is completed, it calls `doneProc`. Do not use `changeEnterFun` without specifying a `doneProc` because it will not return from the original enterfunction until the changed enterfunction returns or is cancelled.

Cadence User Interface SKILL Functions Reference

User Entry Functions

`changeEnterFun` allows a union of all the enterfunction arguments, although it uses only the arguments that apply to the enterfunction specified by `s_enterFun`.

Arguments

<code>s_enterFun</code>	Symbol for the enterfunction you wish to change to.
<code>l_promptList</code>	List of prompt strings used by the new enterfunction.
<code>l_pointList</code>	List of points that can be preloaded for the new enterfunction to operate on.
<code>x_pointLimit</code>	Number of points needed by the new enterfunction.
<code>s_form</code>	Form supplied to the new enterfunction.
<code>t_addProcName</code>	<code>addPointProc</code> corresponding to the new enterfunction.
<code>t_delProcName</code>	<code>delPointProc</code> corresponding to the new enterfunction.
<code>t_initProcName</code>	<code>initProc</code> corresponding to the new enterfunction.
<code>t_doneProcName</code>	<code>doneProc</code> corresponding to the new enterfunction.
<code>t_formProcName</code>	<code>formProc</code> corresponding to the new enterfunction.
<code>t_pathStyle</code>	PathStyle to use if the new enterfunction is <code>enterPath</code> .
<code>f_pathWidth</code>	Path width to use if the new enterfunction is <code>enterPath</code> .
<code>f_beginExtent</code>	Beginning extension to use if the new enterfunction is <code>enterPath</code> .
<code>f_endExtent</code>	Ending extension to use if the new enterfunction is <code>enterPath</code> .
<code>g_dontDraw</code>	Either <code>t</code> or <code>nil</code> to determine whether or not to draw rubber band lines.
<code>g_alwaysMap</code>	Either <code>t</code> or <code>nil</code> to determine whether or not to immediately display the options form.
<code>g_acceptString</code>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept string input without terminating the enterfunction.

Cadence User Interface SKILL Functions Reference

User Entry Functions

<i>g_acceptNumber</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to accept numeric input without terminating the enterfunction.
<i>g_noInfix</i>	Either <code>t</code> or <code>nil</code> to determine whether or not to disable infix mode.
<i>t_cmdName</i>	The name of the new enterfunction. You can use this name as an identifier to control nesting.

Value Returned

<code>t</code>	Returns <code>t</code> if the original enterfunction was successfully terminated and the new enterfunction successfully initialized.
<code>nil</code>	Returns <code>nil</code> if the enterfunction is not successfully terminated or the new enterfunction was not initialized.

Example

```
procedure( circleDone( w done pts )
  if( done then
    printf("Circle = %L\n" pts )
  else
    println("Circle entry aborted.")
  )
)

procedure( ellipseDone( w done pts )
  if( done then
    printf("Ellipse = %L\n" pts )
  else
    println("Ellipse entry aborted.")
  )
)

enterCircle( ?prompts list( "Center" "Radius")
  ?doneProc "circleDone" ?form circleForm ?alwaysMap t)
```

`circleForm` is a form containing options for drawing a circle. Since the `alwaysMap` flag is `t`, the form is displayed. The prompt in the CIW reads “Center.” Enter point *p1*. The prompt reads “Radius,” and a rubberbanding circle with center at *p1* follows the cursor.

```
changeEnterFun( 'enterEllipse
  ?prompts list( "First Corner" "Second Corner")
  ?doneProc "ellipseDone" ?form ellipseForm ?alwaysMap t)
```

The message “Circle entry aborted” is printed out, and the circle options form and the rubberbanding circle disappear. The prompt in the CIW is changed to “First corner” and the `ellipseForm` appears. Enter *p2*, and a rubberbanding ellipse is drawn.

Cadence User Interface SKILL Functions Reference

User Entry Functions

changeNextEnterFun

```
changeNextEnterFun(  
    [w_window]  
)  
=> t / nil
```

Description

If an enterfunction is currently active, sets a flag that causes the next enterfunction invocation for the current window or a specified window to use the `changeEnterFun` logic rather than being nested.

Arguments

<code>w_window</code>	Window where the <code>changeEnterFun</code> logic is to be used. If no argument is specified, the current window is used.
-----------------------	--

Value Returned

<code>t</code>	Returns <code>t</code> if the flag is set.
<code>nil</code>	Returns <code>nil</code> if the flag is not set.

Reference

[clearAllEnterFunctions](#), [changeEnterFun](#), [changeNextEnterFun](#), [hiMarkNonNestable](#), [hiMarkNestable](#)

Cadence User Interface SKILL Functions Reference

User Entry Functions

clearAllEnterFunctions

```
clearAllEnterFunctions( )  
=> t / nil
```

Description

Cancels all nested enterfunctions associated with the current window.

Same as calling `cancelEnterFun` repeatedly until no more enterfunctions are running in the current window.

Arguments

None.

Value Returned

<code>t</code>	Returns <code>t</code> if the functions are canceled or if there is nothing to cancel.
<code>nil</code>	Returns <code>nil</code> if there are no windows open other than the Command Interpreter Window.

Reference

[`changeNextEnterFun`](#), [`changeEnterFun`](#), [`hiMarkNonNestable`](#), [`hiMarkNestable`](#)

Cadence User Interface SKILL Functions Reference

User Entry Functions

hiGetCurrentCmd

```
hiGetCurrentCmd(  
    w_window  
)  
=> t_commandName / nil
```

Description

Returns the command name associated with the enterfunction currently running in a window, or `nil` if no enterfunction is running.

Arguments

<i>w_window</i>	Window ID.
-----------------	------------

Value Returned

<i>t_commandName</i>	Command name associated with the enterfunction running in the window.
<i>nil</i>	Returns <code>nil</code> if an enterfunction is not running in the window.

Cadence User Interface SKILL Functions Reference

User Entry Functions

hiInEnterFun

```
hiInEnterFun(  
    [w_wtype]  
)  
=> s_enterFunction / nil
```

Description

Returns the symbol of the enterfunction pending in the window.

Arguments

<i>w_wtype</i>	Window ID. If you do not specify this argument, the current window is used.
----------------	---

Value Returned

<i>s_enterFunction</i>	Symbol of the enterfunction pending in the window. For example, 'enterPoints.
<i>nil</i>	No enterfunction is pending in the window.

Cadence User Interface SKILL Functions Reference

User Entry Functions

hiMarkNestable

```
hiMarkNestable(  
    g_symbolOrList  
)  
=> t / nil
```

Description

Makes enterfunctions nestable. When an enterfunction is running and a new one is called, the new enterfunction replaces the old enterfunction in the stack. When the new enterfunction is finished or canceled, the old enterfunction is resumed. This function reverses the effect of `hiMarkNonNestable`.

Arguments

<i>g_symbolOrList</i>	Symbol or list of symbols representing one or more enterfunctions.
-----------------------	--

Return Value

<i>t</i>	Returns <i>t</i> if the functions were made nestable.
<i>nil</i>	Returns <i>nil</i> if there was an error.

Reference

[hiMarkNonNestable](#), [changeEnterFun](#), [changeNextEnterFun](#), [clearAllEnterFunctions](#)

Cadence User Interface SKILL Functions Reference

User Entry Functions

hiMarkNonNestable

```
hiMarkNonNestable(  
    g_symbolOrList  
)  
=> t / nil
```

Description

Makes enterfunctions non-nestable. When an enterfunction is running and a new one is called, the new enterfunction replaces it in the stack. When the new enterfunction is finished or canceled, the enterfunction it replaced will not be returned to, but will have been canceled by the new enterfunction.

Arguments

<i>g_symbolOrList</i>	Symbol or list of symbols representing one or more functions, presumably enterfunctions.
-----------------------	--

Value Returned

<i>t</i>	Returns <i>t</i> if the operation succeeded.
<i>nil</i>	Returns <i>nil</i> if the operation failed.

Reference

[changeNextEnterFun](#), [clearAllEnterFunctions](#), [hiMarkNestable](#)

Cadence User Interface SKILL Functions Reference

User Entry Functions

hiUpdate

```
hiUpdate( )  
=> t
```

Description

Updates window prompts and application-specific state variables. Especially important if an enterfunction has been canceled in a window that is not the current window.

Arguments

None.

Value Returned

t Returns t after the update is finished.

Cadence User Interface SKILL Functions Reference

User Entry Functions

hiToggleEnterForm

```
hiToggleEnterForm(  
    [g_form]  
)  
=> t / nil
```

Description

Displays or removes the options form of the enterfunction in the current window.

If the options form was not shown, it is shown; if the options form is shown, it is removed. If the enterfunction does not have an options form, or if no enterfunction is active, a warning dialog box is displayed.

Arguments

<i>g_form</i>	Options form to toggle on or off. This allows you to toggle an enterfunction form regardless of whether it is associated with the current window.
---------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the display status of the form was changed.
<i>nil</i>	Returns <i>nil</i> if the display status of the form was not changed.

Example

```
enterCircle( ?prompts list( "Center" "Radius" ) ?form circleForm )
```

circleForm is a form you have created that contains options for your circle. Now type

```
hiToggleEnterForm( circleForm )
```

The form pops up, and you can set options. Type it again and the form disappears again.

Cadence User Interface SKILL Functions Reference

User Entry Functions

undrawEnterFun

```
undrawEnterFun(  
    )  
=> t / nil
```

Description

Undraws any rubberbanding shapes. Called when an enterfunction is active. If you enter data in the window, rubberbanding resumes.

Arguments

None.

Value Returned

`t` Returns `t` if the undraw is successful.

`nil` Returns `nil` if there was no enterfunction pending in the current window.

Example

```
enterPolygon( ?prompts list("First" "Next")  
    ?pointlist list( p1 p2 p3 ) )
```

The prompt “Next” appears in the CIW, and line segments connecting *p1* and *p2*, and *p2* and *p3* appear on the screen.

```
undrawEnterFun( )
```

The segments are erased, but the prompt remains signifying that the enterfunction is still active.

Cadence User Interface SKILL Functions Reference

User Entry Functions

drawEnterFun

```
drawEnterFun(  
    )  
=> t / nil
```

Description

Redraws rubberbanding shapes. Called when an enterfunction is active.

Arguments

None.

Value Returned

<code>t</code>	Returns <code>t</code> if the draw is successful.
<code>nil</code>	Returns <code>nil</code> if there is no enterfunction pending in the current window.

Example

After following the example on the previous page, type

```
drawEnterFun( )
```

The two segments connecting the three points are redrawn.

Cadence User Interface SKILL Functions Reference
User Entry Functions

Encapsulation Window

The following topics are discussed in this chapter:

- [Overview](#) on page 634
- [hiEncap](#) on page 635
- [hiSetEncapSkillCmd](#) on page 637
- [hiGetEncapSkillCmd](#) on page 639
- [hiSetEncapPrompt](#) on page 640
- [hiSetEncapHistory](#) on page 641
- [hiAppendInputCmd](#) on page 643
- [hiFocusToEncap](#) on page 644

Overview

An Encapsulation window is similar to the CIW except that it does not have a mouse binding display.

The optional window manager border contains the title “Encapsulation.” You can replace this name using the SKILL functions [hiSetWindowName](#) or [hiEncap](#). The banner contains pulldown menus providing the default system menu that contains the SKILL [hiCloseWindow](#) function.

Each command line is copied into the output area from the input line as it is accepted for execution. The input command line is then placed with the user-specifiable SKILL command wrapper through the [hiSetEncapSkillCmd](#) command. The SKILL command and any error messages or output produced by that command are displayed in the CIW output area. Only the input history of the encapsulation session is copied into the Encapsulation output window.

The input window behaves like the input window of the CIW.

You control the size and location of the Encapsulation window. The display manager defaults to a certain minimum window size.

The following properties can be set on an encapsulation window:

- `sendAllInput`

When set to `t`, the encapsulation window accepts a single carriage return as input. When this property is not set (or set to `nil`), single carriage returns are ignored by the encapsulation window. (This is the default behavior.) See [hiSetEncapSkillCmd](#).

- `noAutoNewline`

See [hiSetEncapHistory](#).

- `obeySpecialChars`

See [hiAppendInputCmd](#).

The following commands support Encapsulation windows.

Cadence User Interface SKILL Functions Reference

Encapsulation Window

hiEncap

```
hiEncap(  
    [g_boxSpec]  
    [t_cmd]  
    [t_title]  
)  
=> w_window / nil
```

Description

Invokes an encapsulation session by creating and displaying an encapsulation window.

The window created has a widget type of “encap” and an application type of “encap”. (See [hiCreateWindow](#).)

Note: If you get the warning message "Cannot allocate colormap entry for default background," you can ignore it. This message is displayed if you have set CDS_USE_PRIVATE_CMAP to "true" or have used a private colormap. It does not affect the encapsulation window.

Arguments

<i>g_boxSpec</i>	Screen coordinates specifying the lower left and upper right corners of the window (assuming 0:0 to be the lower left corner of the screen). If <i>boxSpec</i> is nil, either you are prompted with a rubberbanding box to enter a window size or a default size is used depending on the value of the global variable <i>hivWinStyle</i> . The 'default and 'interactive symbols are also acceptable values for the <i>boxSpec</i> argument.
<i>t_cmd</i>	SKILL function to which all input sent to the Encapsulation window is redirected. This function acts as a wrapper function to the input command line entered during the current encapsulation session. (See hiSetEncapSkillCmd .)
<i>t_title</i>	String that appears in the window manager banner of the encapsulation window. If not specified, the string "Encapsulation" appears.

Cadence User Interface SKILL Functions Reference

Encapsulation Window

Value Returned

<i>w_window</i>	Opens an encapsulation window and returns the ID of the window.
<i>nil</i>	Returns <i>nil</i> if it cannot open a window.

Example

```
hiEncap( )  
=> window:2
```

Opens an encapsulation window and returns its *windowId*.

Cadence User Interface SKILL Functions Reference

Encapsulation Window

hiSetEncapSkillCmd

```
hiSetEncapSkillCmd(  
    w_windowId  
    t_skillCmd  
)  
=> t / nil
```

Description

Registers the SKILL function that wraps or embraces the input command line entered during the current encapsulation session.

Any input entered in the input window of the encapsulation window is redirected to the SKILL function `t_skillCmd`. This function should accept only one argument—the string that is the line of input to the encapsulation window.

Note: If the property `sendAllInput` is set to `t` on the encapsulation window, an empty string is passed to the `t_skillCmd` function. If the property is not set, the default action is that single carriage returns are ignored.

Arguments

<code>w_windowId</code>	Encapsulation window ID returned by hiEncap .
<code>t_skillCmd</code>	SKILL function to use as the wrapper function.

Value Returned

<code>t</code>	Returns <code>t</code> if the function is registered.
<code>nil</code>	Returns <code>nil</code> and issues a warning message if the function is not registered.

Example

Create the procedure to be used as the wrapper:

```
procedure(processString(input "t")  
;process the input string typed to the encapsulation  
;window  
;...  
)
```

Cadence User Interface SKILL Functions Reference

Encapsulation Window

Create an encapsulation window:

```
w = hiEncap(list(100:100 500:500))  
hiSetEncapSkillCmd(w "processString")
```

When input is typed in the encapsulation window:

```
countAll
```

the wrapper function is set up, using the encapsulation window input as its argument:

```
processString("countAll")
```

The `processString` function then processes the input string and either redirects it to another process or calls the appropriate function.

Cadence User Interface SKILL Functions Reference

Encapsulation Window

hiGetEncapSkillCmd

```
hiGetEncapSkillCmd(  
    w_window  
)  
=> t_skillCmd
```

Description

Returns the SKILL function associated with a window.

Arguments

<i>w_window</i>	Encapsulation window ID returned from hiEncap .
-----------------	---

Value Returned

<i>t_skillCmd</i>	Returns the SKILL wrapper function string associated with the encapsulation window.
-------------------	---

Example

```
hiGetEncapSkillCmd(w)  
=> "processString"
```

Cadence User Interface SKILL Functions Reference

Encapsulation Window

hiSetEncapPrompt

```
hiSetEncapPrompt(  
    w_window  
    t_prompt  
)  
=> t / nil
```

Description

Sets the prompt for an encapsulation window.

Arguments

<i>w_window</i>	Encapsulation window ID returned by hiEncap .
<i>t_prompt</i>	String to be used as the prompt. The default prompt is the greater than character (>).

Value Returned

<i>t</i>	Returns <i>t</i> if the prompt is set.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the prompt is not set.

Example

```
hiSetEncapPrompt(w "What_now?")
```

Returns *t* and displays the prompt "What_now?" directly above the input line in the encapsulation window.

Cadence User Interface SKILL Functions Reference

Encapsulation Window

hiSetEncapHistory

```
hiSetEncapHistory(  
    w_window  
    t_msg  
    [g_redraw]  
)  
=> t / nil
```

Description

Copies a message into the next available insertion position in an encapsulation output window.

Arguments

<i>w_window</i>	Encapsulation window ID returned from hiEncap .
<i>t_msg</i>	String that you want appended (inserted) to the last line of the output portion of the encapsulation window.
<i>g_redraw</i>	<p>Either <code>t</code> or <code>nil</code> to determine whether a message sent to the encapsulation output window is displayed or saved. If set to <code>nil</code>, messages sent to the encapsulation window are buffered (saved) and are not displayed until a newline character is explicitly specified in <i>t_msg</i>. For performance reasons, this flag should be set when sending long strings to the window. If set to <code>t</code> (the default), all messages sent to the encapsulation window are immediately displayed with a newline automatically appended.</p> <p>Note: If the property <code>noAutoNewline</code> on the encapsulation window is set to <code>t</code>, the newline is not automatically appended to the <i>t_msg</i> string, and a <code>'\n'</code> character must be explicitly specified in the string before the string is sent and displayed to the window output.</p>

Value Returned

<i>t</i>	Returns <i>t</i> if the string is copied to the encapsulation output window.
----------	--

Cadence User Interface SKILL Functions Reference

Encapsulation Window

`nil`

Returns `nil` and issues a warning message if the string is not copied to the encapsulation output window.

hiAppendInputCmd

```
hiAppendInputCmd(  
    w_window  
    t_cmd  
)  
=> t / nil
```

Description

Appends a command to the input area of an encapsulation window.

If the encapsulation window has the property `obeySpecialChars` set to `t`, this function treats the character `'\n'` in `t_cmd` as a newline (equivalent to pressing the carriage return key in the encapsulation window). If this property is not set, the `'\n'` and any characters following this in `t_cmd` are ignored.

Arguments

<code>w_window</code>	Encapsulation window ID returned from hiEncap .
<code>t_cmd</code>	SKILL function string to append to the input area.

Value Returned

<code>t</code>	Returns <code>t</code> if the string is appended to the input area.
<code>nil</code>	Returns <code>nil</code> and issues a warning message if the window specified by <code>w_window</code> is invalid or if the function <code>t_cmd</code> is null.

Example

```
w = hiEncap(list(100:100 500:500) "" "encap")  
w->obeySpecialChars = t  
hiAppendInputCmd(w "ab")      ; input has 'ab'  
hiAppendInputCmd(w "cd")      ; input has 'abcd'  
hiAppendInputCmd(w "\n")      ; sends 'abcd' to output  
hiAppendInputCmd(w "ab\ncd\nef")  
    ; sends 'ab' to output  
    ; sends 'cd' to output  
    ; input has 'ef'  
hiAppendInputCmd(w "gh\n")    ; sends 'efgh' to output
```

Cadence User Interface SKILL Functions Reference

Encapsulation Window

hiFocusToEncap

```
hiFocusToEncap(  
    w_encap  
    [w_window]  
)  
=> t / nil
```

Description

Temporarily focuses the keyboard input to an encapsulation window.

The keyboard focus to the encapsulation window is maintained until a carriage return is entered or until the pointer focus is moved from one window into another.

Arguments

<i>w_encap</i>	Encapsulation window ID returned from hiEncap .
<i>w_window</i>	Window from which the keyboard focus is being transferred. If not specified, the current window is used.

Value Returned

<i>t</i>	Returns <i>t</i> if the focus is transferred to the encapsulation window.
<i>nil</i>	Returns <i>nil</i> and issues an error message if the focus is not transferred.

Viewfile Window

The set of viewfile window functions described in this chapter provides a consistent way to manipulate text view file windows.

The following topics are discussed in this chapter:

- [Viewfile Management](#) on page 646
- [Setting the Highlight Color](#) on page 677
- [Word Delimiters](#) on page 683
- [Miscellaneous](#) on page 688

Viewfile Management

A text viewfile window (also known as a *showFile* window) is a rectangular area on the screen that lets you view textual output. The following functions help you work within a viewfile window. Viewfiles allow you to choose a file to work with, save a file under a different name, find a particular text string, and maneuver within the view file window using various criteria. You can also work with a selected class and enable or disable updating from a changing file.

The functions described in this section pertain to viewfile windows only. These windows must have a widget type of “text” and should have been created either from a call to `view` or by a call to `hiCreateWindow` with the widget type set to `text`.

Cadence User Interface SKILL Functions Reference

Viewfile Window

view

```
view(  
    t_file  
    [g_boxSpec]  
    [g_title]  
    [g_autoUpdate]  
    [l_iconPosition]  
)  
=> w_windowId / nil
```

Description

Creates a viewfile window, inserts the banner menu, sets the window and icon names, and displays the specified file in the window. `view` is a wrapper function.

Arguments

<i>t_file</i>	File name to view. This is the only required argument for this function.
<i>g_boxSpec</i>	Screen coordinates of the lower left and upper right corners of the window (assuming 0:0 to be the lower left corner of the screen). If <i>g_boxSpec</i> is <code>nil</code> , either you are prompted with a rubberbanding box to enter a window size, or a default size is used, depending on the value of the global variable <code>hivWinStyle</code> . The 'default' and 'interactive' symbols are also acceptable values for the <i>g_boxSpec</i> argument. If <i>g_boxSpec</i> is a window ID, the widget type of the window is changed to <code>text</code> , if it isn't already.
<i>g_title</i>	Title for the window. This title appears in the window manager banner of the viewfile window. If not specified, the file name is used.
<i>g_autoUpdate</i>	Either <code>t</code> or <code>nil</code> to determine whether continuous updating of the display from the file is desired. If not specified or <code>nil</code> , the display does not reflect changes made by appending to the end of the file. If <code>t</code> , the display is updated when text is appended to the end of the file. Changes to the file other than appending to the end are not displayed. Setting <i>g_autoUpdate</i> to <code>t</code> is equivalent to calling <code>hiEnableTailViewfile</code> .

Cadence User Interface SKILL Functions Reference

Viewfile Window

l_iconPosition This argument is passed to `hiCreateWindow`. For more information, see [hiCreateWindow](#).

Value Returned

w_windowId Returns the window ID of the created viewfile window.

nil Issues an error message and returns `nil` if the viewfile window is not created.

hiSetViewfile

```
hiSetViewfile(  
    w_windowId  
    t_fileName  
)  
=> t / nil
```

Description

Reads a file and displays its contents in a viewfile window that has already been created.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_fileName</i>	Name of the file to be read and displayed

Value Returned

<i>t</i>	Returns <i>t</i> if the file is read and displayed.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid, if <i>t_fileName</i> is not found, or if <i>t_fileName</i> is null.

Cadence User Interface SKILL Functions Reference

Viewfile Window

hiSaveViewfile

```
hiSaveViewfile(  
    [w_windowId]  
    [t_fileName]  
    [g_donotOverwrite]  
)  
=> t / nil
```

Description

Saves the contents of the viewfile window to a file on disk.

Arguments

<i>w_windowId</i>	Window you want the function to act on. The default is the current window.
<i>t_fileName</i>	Name of the saved file.
<i>g_donotOverwrite</i>	If set to <code>t</code> , it does not save the file if the file name already exists. If set to <code>nil</code> , it will overwrite the file.

Value Returned

<code>t</code>	Returns <code>t</code> if the file is saved.
<code>nil</code>	Returns <code>nil</code> if <i>w_windowId</i> is invalid, if <i>t_fileName</i> is not found, or if <i>t_fileName</i> is null.

Cadence User Interface SKILL Functions Reference

Viewfile Window

hiSaveAsViewfile

```
hiSaveAsViewfile(  
    [w_windowId]  
    [t_fileName]  
)  
=> t / nil
```

Description

Saves the contents of the current window to the specified name.

This function does not save the file if the file name already exists. You must specify a unique file name before the file is saved.

Arguments

<i>w_windowId</i>	Window you want the function to act on. The default is the current window.
<i>t_fileName</i>	Unique name for the saved file.

Value Returned

<i>t</i>	Returns <i>t</i> if the file is saved.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid, if <i>t_fileName</i> already exists, or if <i>t_fileName</i> is null.

hiStartGenTextIndex

```
hiStartGenTextIndex(  
    w_windowId  
    t_text  
)  
=> x_index / nil
```

Description

Searches the specified text string from the beginning of the file and returns the first character position of the matching text.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_text</i>	Text string to search for.

Value Returned

<i>x_index</i>	Returns the first character position of the matching text if the text is found.
nil	Returns nil if <i>w_windowId</i> is invalid, if <i>t_text</i> is not found, or if <i>t_text</i> is null.

hiGenTextIndex

```
hiGenTextIndex(  
    w_windowId  
    t_text  
    x_fromIndex  
)  
=> x_index / nil
```

Description

Returns the character position that is the next occurrence of the text string relative to a specified position in the file.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_text</i>	Text string to search for.
<i>x_fromIndex</i>	Character position to start the search from.

Value Returned

<i>x_index</i>	Returns the character position of the matching text.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid, if <i>t_text</i> is not found or null, or no more occurrences of <i>t_text</i> are found.

hiGetTextSelection

```
hiGetTextSelection(  
    w_windowId  
    [x_class]  
)  
=> l_text / nil
```

Description

Retrieves the currently selected text strings from within a specified class.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_class</i>	The group class that the selections should belong to. Possible values range from 0 to <code>MAX_INT</code> . Zero is the default.

Value Returned

<i>l_text</i>	Returns a list of all the selected text strings of the class specified.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid or if no text is currently selected for the specified class.

hiGetTextSelByLoc

```
hiGetTextSelByLoc(  
    w_windowId  
    [x_class]  
)  
=> l_locationPair / nil
```

Description

Returns a list of starting and ending location pairs of all selected text strings in the specified class.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>l_locationPair</i>	Returns a list of the beginning and ending locations of all selected text strings of the class specified.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid or if no text is currently selected.

hiSetTextSelection

```
hiSetTextSelection(  
    w_windowId  
    t_text  
    [x_class]  
)  
=> t / nil
```

Description

Highlights the next occurrence of a specified string relative to the current cursor position. If no matching string is found between the current cursor position and the end of the file, it wraps around to the beginning of the file and continues the search.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_text</i>	Text string to search for.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>t</i>	Returns <i>t</i> if a match is found.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid, if <i>t_text</i> is not found, or if <i>t_text</i> is null.

hiSelectTextByLoc

```
hiSelectTextByLoc(  
    w_windowId  
    l_locationPair  
    [x_class]  
)  
=> t / nil
```

Description

Highlights a list of strings specified as a list of starting and ending location pairs.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>l_locationPair</i>	List of starting and ending location pairs for the selection.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>t</i>	Returns <i>t</i> if a match is found.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid or if <i>l_locationPair</i> is out of range.

hiSetTextSelectAll

```
hiSetTextSelectAll(  
    w_windowId  
    t_text  
    [x_class]  
)  
=> t / nil
```

Description

Highlights all occurrences of a specified string and resets the cursor to the top of the file.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_text</i>	Text string to search for.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>t</i>	Returns <i>t</i> if a match is found.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid, if <i>t_text</i> is not found, or if <i>t_text</i> is null.

Cadence User Interface SKILL Functions Reference

Viewfile Window

hiUnselectText

```
hiUnselectText(  
    w_windowId  
    t_text  
    [x_class]  
)  
=> t / nil
```

Description

Unhighlights the next occurrence of selected text in the viewfile relative to the cursor position.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_text</i>	Text string to be unselected.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>t</i>	Returns <i>t</i> if a match is found.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiUnselectTextByLoc

```
hiUnselectTextByLoc(  
    w_windowId  
    l_locationPair  
    [x_class]  
)  
=> t / nil
```

Description

Unhighlights selected text specified as a list of starting and ending location pairs in the given window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>l_locationPair</i>	List of starting and ending location pairs.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>t</i>	Returns <i>t</i> if the text is unselected.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiUnselectTextClass

```
hiUnselectTextClass(  
    w_windowId  
    x_class  
)  
=> t / nil
```

Description

Unhighlights all occurrences of selected text belonging to the specified class.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>t</i>	Returns <i>t</i> if the text is unhighlighted.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid or if no selected text matches with the class.

hiUnselectTextAll

```
hiUnselectTextAll
    w_windowId
    [t_text]
    [x_class]
)
=> t / nil
```

Description

Unselects (unhighlights) all occurrences of the selected text that match with the specified pattern *t_text* and class *x_class* in the viewfile. If the search pattern *t_text* is not specified, unselects all occurrences of the selected text of all classes. If the class *x_class* is not specified, unselects all occurrences of the selected text that match with the specified pattern *t_text*. Afterwards, the cursor position is reset to the top of the file. If an empty string is passed for *t_text*, and *x_class* is specified, unselects all strings selected in that class

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_text</i>	Text to be unselected.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value.

Value Returned

<i>t</i>	Returns <i>t</i> if the text is unselected.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid, if no selected text matches with the specified text and class, or if nothing is selected.

hiScrollWindowLeft

```
hiScrollWindowLeft(  
    w_windowId  
)  
=> t / nil
```

Description

Scrolls the window left one screenful of text at a time.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window is scrolled.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiScrollWindowRight

```
hiScrollWindowRight(  
    w_windowId  
)  
=> t / nil
```

Description

Scrolls the window right one screenful of text at a time.

Arguments

<code>w_windowId</code>	Window you want the function to action.
-------------------------	---

Value Returned

<code>t</code>	Returns <code>t</code> if the window is scrolled.
<code>nil</code>	Issues a warning and returns <code>nil</code> if <code>w_windowId</code> is invalid.

hiScrollWindowUp

```
hiScrollWindowUp(  
    w_windowId  
)  
=> t / nil
```

Description

Scrolls the window up one screenful of text at a time.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window is scrolled.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiScrollWindowDown

```
hiScrollWindowDown(  
    w_windowId  
)  
=> t / nil
```

Description

Scrolls the window down one screenful of text at a time.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window is scrolled.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiScrollWindowTop

```
hiScrollWindowTop(  
    w_windowId  
)  
=> t / nil
```

Description

Scrolls the window to the top position of the viewable window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window is scrolled.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiScrollWindowBottom

```
hiScrollWindowBottom(  
    w_windowId  
)  
=> t / nil
```

Description

Scrolls the window to the bottom position of the viewable window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window is scrolled.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiScrollWindowToCurrentIndex

```
hiScrollWindowToCurrentIndex(  
    w_windowId  
)  
=> t / nil
```

Description

Scrolls the window to the current cursor index position in the viewable window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window is scrolled.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiScrollWindowToIndex

```
hiScrollWindowToIndex(  
    w_windowId  
    x_index  
)  
=> t / nil
```

Description

Scrolls the window to the index position in the specified window and forces the text at the given position to be displayed.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_index</i>	Zero-relative character position to be displayed.

Value Returned

<i>t</i>	Returns <i>t</i> if the text is displayed.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> or <i>x_index</i> is invalid.

Cadence User Interface SKILL Functions Reference

Viewfile Window

hiGetCurrentIndex

```
hiGetCurrentIndex(  
    w_windowId  
)  
=> x_index / nil
```

Description

Returns the current cursor index position of text relative to 0:0.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>x_index</i>	Returns the current cursor position.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiSetCurrentIndex

```
hiSetCurrentIndex(  
    w_windowId  
    x_index  
)  
=> t / nil
```

Description

Sets the current cursor index position of the text in a window.

The current index position is modified by hiTextDisplayString.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_index</i>	Specifies the zero-relative character position of the cursor.

Value Returned

<i>t</i>	Returns <i>t</i> if the cursor position is set.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> or <i>x_index</i> is invalid.

hiDisableTailViewfile

```
hiDisableTailViewfile(  
    w_windowId  
)  
=> t / nil
```

Description

Turns off the ability to repeatedly read from the end of a file.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the capability is turned off.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

Cadence User Interface SKILL Functions Reference

Viewfile Window

hiEnableTailViewfile

```
hiEnableTailViewfile(  
    w_windowId  
)  
=> t / nil
```

Description

Turns on the ability to repeatedly read from the end of a file and update the viewfile window. This will continually update the viewfile window with text that is appended to the end of the file being viewed. This only works when appending to a file. If the file is modified in another way (such as edited using an editor), the results are unpredictable. This function is equivalent to the Unix command `tail_f` and works the same way.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the capability is turned on.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiGetTextClass

```
hiGetTextClass(  
    w_windowId  
)  
=> x_class / nil
```

Description

Returns the value of the current active text class. If hiSetTextClass() has not been set, the current active text class will be 0. Also, a few other functions may reset the active text class back to 0 or possibly another value. Otherwise the currently active text class will be the one set by the last call to hiSetTextClass().

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>x_class</i>	Returns the current active text class.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiSetTextClass

```
hiSetTextClass(  
    w_windowId  
    x_class  
)  
=> t / nil
```

Description

Sets the active selection (highlight) text class in a window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_class</i>	Group of selections. Values range from 0 to MAX_INT. Zero is the default value and uses the window inverse color for highlighting selected text. The highlight selection color for class values of 1 through 25 can be set using the function <u>hiSetTextHighlightColor()</u> . Class values greater than 25 use the same highlight colors as class 0.

Value Returned

<i>t</i>	Returns <i>t</i> if the class is set.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

Setting the Highlight Color

The following functions let you set the highlight color, update the text selection colors, refresh the viewfile window, and display text in a particular way.

hiSetTextHighlightColor

```
hiSetTextHighlightColor(  
    x_class  
    l_foreground  
    l_background  
)  
=> t / nil
```

Description

Sets the foreground and background highlight colors of a class to the specified RGB value.

The range of the class is from 1 through 25. Only classes 1 through 10 are initialized at startup. Whenever a class number is used from 11 through 25 that has not had its color initialized with this function, the highlight color will be the same as for class 0. Since 25 is the highest class number that can have a unique color combination assigned, any class above 25 will always use the same highlight color as for class 0.

The colors can be specified either as actual RGB values with ranges from 0 to 1000 or as the return value of *nameToColor* which translates the predefined color (as contained in the server's rgb.txt file) into its corresponding RGB values. For example, the following command highlights class 5 with white as the foreground color and navy as the background color.

```
hiSetTextHighlightColor(5 nameToColor("white") nameToColor("navy"))
```

Following are the default settings for classes 1 through 10. Class 0 uses the background color as the foreground (text) color and the text (foreground) color as the background color for selections:

Class	Foreground	Background
Number	Color	Color
1	white	green
2	white	red
3	yellow	navy
4	red	yellow
5	white	blue
6	white	violet
7	black	turquoise
8	yellow	orange

Cadence User Interface SKILL Functions Reference

Viewfile Window

Class	Foreground	Background
9	yellow	magenta
10	white	black

Arguments

<i>x_class</i>	Class number for selection. Valid range is 1 through 25. Zero is the default value and uses the window inverse color.
<i>l_foreground</i>	Color for foreground.
<i>l_background</i>	Color for background.

Value Returned

<i>t</i>	Returns <i>t</i> if the classes have been set.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>x_class</i> is smaller than one or greater than 25 or if the RGB value of the foreground and background color is not within the range of 0 to 1000.

hiTextDisplayString

```
hiTextDisplayString(  
    w_windowId  
    t_text  
    g_erase  
    [g_dontScrollTop]  
)  
=> t / nil
```

Description

Displays a text string in a window.

Note: The current cursor index and the corresponding cursor position are modified by this function.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_text</i>	Text string to display.
<i>g_erase</i>	Either <i>t</i> or <i>nil</i> to determine how the string should be displayed. If <i>g_erase</i> is <i>t</i> , the content of the window is erased and the text string is displayed beginning at the first character position in the window. If <i>g_erase</i> is <i>nil</i> , the text string is displayed following the last character position of the current text.
<i>g_dontScrollTop</i>	Either <i>t</i> or <i>nil</i> to determine whether the window should be scrolled. If <i>g_dontScrollTop</i> is not specified or is <i>nil</i> , the window is scrolled back to display the first character position of the window, and the current cursor index position is set to zero. If <i>g_dontScrollTop</i> is <i>t</i> , the window is scrolled to display the last character position of the window, and the current cursor index position is set to the last character position of the window.

Value Returned

<i>t</i>	Returns <i>t</i> if the string is displayed.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid or if <i>t_text</i> is null.

Cadence User Interface SKILL Functions Reference

Viewfile Window

hiRefreshTextWindow

```
hiRefreshTextWindow(  
    w_window  
)  
=> t / nil
```

Description

Redraws all the selections and refreshes the viewfile window.

Arguments

<i>w_window</i>	Window you want the function to act on.
-----------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the window is refreshed.
<i>nil</i>	Returns <i>nil</i> if <i>w_window</i> is invalid.

hiUpdateTextSelectionColors

```
hiUpdateTextSelectionColors(  
    w_windowId  
)  
=> t / nil
```

Description

Redraws all selections visible in the window and updates any colors previously reset with [hiSetTextHighlightColor](#).

After resetting the highlight color using [hiSetTextHighlightColor](#), this function visually updates the colors of the selections.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

t	Returns t if the selections are redrawn.
nil	Returns nil if <i>w_windowId</i> is invalid.

Word Delimiters

The word delimiter functions that follow let you customize the list of word delimiters.

hiGetTextWordDelimiter

```
hiGetTextWordDelimiter(  
    w_windowId  
)  
=> l_delimiters / nil
```

Description

Returns the current settings of the word delimiter list for the window.

The default word delimiters are defined as any character not alphanumeric.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>l_delimiters</i>	Returns a list of characters currently set as word delimiters.
<i>nil</i>	Returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiAddTextWordDelimiter

```
hiAddTextWordDelimiter(  
    w_windowId  
    t_delimiters  
)  
=> t / nil
```

Description

Adds characters to the word delimiter list for the window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_delimiters</i>	Characters to be added to the word delimiter list.

Value Returned

<i>t</i>	Returns <i>t</i> if the characters are added.
<i>nil</i>	Issues a warning and returns <i>nil</i> if one of the characters is already defined as a word delimiter, if <i>t_delimiters</i> is <i>NULL</i> or an empty string, or if <i>w_windowId</i> is invalid.

hiRemoveTextWordDelimiter

```
hiRemoveTextWordDelimiter(  
    w_windowId  
    t_delimiters  
)  
=> t / nil
```

Description

Removes characters from the word delimiter list for the window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_delimiters</i>	Characters to be removed from the word delimiter list.

Value Returned

<i>t</i>	Returns <i>t</i> if the characters are removed.
<i>nil</i>	Issues a warning and returns <i>nil</i> if the character in <i>t_delimiters</i> is not found in the word delimiter list, if <i>t_delimiters</i> is <code>NULL</code> or an empty string, or if <i>w_windowId</i> is invalid.

hiReplaceTextWordDelimiter

```
hiReplaceTextWordDelimiter(  
    w_windowId  
    t_delimiters  
)  
=> t / nil
```

Description

Replaces the original word delimiters for the window with a new set of characters.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>t_delimiters</i>	New set of characters to be used as word delimiters.

Value Returned

<i>t</i>	Returns <i>t</i> if the set is replaced.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>t_delimiters</i> is NULL or an empty string, or if <i>w_windowId</i> is invalid.

Miscellaneous

The following functions let you retrieve text at a specific location, get text source length, retrieve a line and column number, a text index, or a display location, and disable the text selection default.

hiGetTextCharAtLoc

```
hiGetTextCharAtLoc(  
    w_windowId  
    x_location  
)  
=> c_text / nil
```

Description

Returns the character of the specified character index location for a window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_location</i>	Character index location. If not specified, the current cursor index location is the default.

Value Returned

<i>c_text</i>	Returns the character at the specified location.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid or if the specified location is out of range.

Cadence User Interface SKILL Functions Reference

Viewfile Window

hiGetTextSourceLength

```
hiGetTextSourceLength(  
    w_windowId  
)  
=> x_length / nil
```

Description

Returns the length of the source in a window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>x_length</i>	Returns the length of the text.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiGetTextLineColumn

```
hiGetTextLineColumn(  
    w_windowId  
    x_location  
)  
=> l_lineColumn / nil
```

Description

Returns the line and column number for an index location in a window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_location</i>	Character index location in which you are interested .

Value Returned

<i>l_lineColumn</i>	Returns a list of the line and column numbers.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid, or if the character location is out of range.

hiGetTextIndexLoc

```
hiGetTextIndexLoc(  
    w_windowId  
    x_line  
    x_column  
)  
=> x_location / nil
```

Description

Returns the character index location of a line and column number in a window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
<i>x_line</i>	Line number.
<i>x_column</i>	Column number.

Value Returned

<i>x_location</i>	Returns the character location.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid, or if the line number or column number is out of range.

hiGetTextDispLoc

```
hiGetTextDispLoc(  
    w_windowId  
)  
=> l_topBottom / nil
```

Description

Returns the top and bottom character locations of the current visible region of a window.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>l_topBottom</i>	Returns a list that contains the top and bottom character locations.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid.

hiDisableTextSelDefault

```
hiDisableTextSelDefault(  
    w_windowId  
)  
=> t / nil
```

Description

Disables the default multi-click selection.

The application should register its own text selection trigger before disabling the default selection. This function is used only when an application needs more control of the interactive text selection. For instance, when doing double-click selection, the application uses its own parser to parse a word and then select the word.

Arguments

<i>w_windowId</i>	Window you want the function to act on.
-------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the default text selection trigger is disabled.
<i>nil</i>	Issues a warning and returns <i>nil</i> if <i>w_windowId</i> is invalid or if the user's text selection trigger is not registered yet.

Display Lists

The following topics are discussed in this appendix:

- [Display List Objects](#) on page 696
- [Creating a Display List](#) on page 697
- [Adding Objects to the Display List](#) on page 700
- [Display List Draw Functions](#) on page 716
- [Display List Pens](#) on page 728
- [Viewing a Display List](#) on page 737
- [A Sample Display List](#) on page 759

Display List Objects

Display lists are used to construct and display pictures within the Design Framework II user interface. display lists are collections of shapes displayed within a form, menu, button, or window. Additionally, display lists are used by the DAG Browser to display pictures.

Display lists use “pens” to hold the graphical properties a shape is drawn with. A pen stores color, line style, and fill style. Pens are referred to by number. When you add a shape to a display list, you specify which pen number it should use.

Display lists are built in two ways:

- By calling the SKILL functions to add objects to the display list.
- By drawing the picture in the icon editor. You can save the picture in a file and load it whenever you need the display list again.

The display list is composed of the following objects:

- Points
- Segments
- Boxes
- Arcs
- Circles
- Donuts
- Polygons
- Paths
- RasterText
- StrokeText
- SkillObjects
- EventObjects

Display lists are not hierarchical in that a display list cannot include another display list.

Coordinates are Cartesian and specified using integer numbers. The range of the integers can be anything you wish. The display list is scaled to fit into the space provided.

Creating a Display List

Display lists are created with `dlMakeDisplayList`.

Cadence User Interface SKILL Functions Reference

Display Lists

dlMakeDisplayList

```
dlMakeDisplayList(  
    )  
=> w_displayList / nil
```

Description

Creates an empty display list you can add objects to.

Value Returned

Returns the display list, or `nil` if an error occurred.

dISetClearOnDraw

```
dISetClearOnDraw(  
    w_dlist  
    x_widgetId  
    g_draw  
    ) => t/nil
```

Description

Specifies whether the area the display list is drawn in is cleared before the display list is drawn.

Arguments

<i>w_dlist</i>	The display list.
<i>x_widgetId</i>	The widget the display list is attached to.
<i>g_draw</i>	<i>t</i> or <i>nil</i> . Specify <i>t</i> if you want the area to be cleared before the display list is drawn; <i>nil</i> otherwise.

Adding Objects to the Display List

Objects are added to the display list in the following manner:

```
dlAddObject(w_dlist x_penNumber..... [s_tagSymbol])  
=> t / nil
```

The first two arguments are always the same. *w_dlist* is the display list the object should be added to, and *x_penNumber* is the number of the pen the object will use when it is drawn.

Pens are collected into a group called a pen table. If you use the default pen table, then there are 64 pens available to you. The pen numbers start at 0; therefore, if you use the default pen table, you can use numbers ranging from 0 to 63. The *s_tagSymbol* is an optional symbol given so you can refer to this shape again. It is always the final argument to a *dlAdd* function. If you define your own pen table, then you can use fewer or more pens; the more pens you create, the more memory is required.

The coordinates are specified using integer numbers and can be any range you choose. All coordinates specified assume 0:0 to be the lower-left corner of the display list object. The display list is automatically scaled to fit into a given area, much like a *geFull* function does with a database object. You should, however, try to choose your coordinate space so it isn't too small. This can avoid integer round-off when converting from display list coordinates to screen coordinates.

All *dlAddObject* functions return *t* on success, and *nil* on failure. Conditions for failure are either an invalid *dlist* argument, or a pen number outside a valid range.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddArc

```
dlAddArc(  
    w_dlist  
    x_penNumber  
    l_point1  
    l_point2  
    n_startAngle  
    n_sweepAngle  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds an arc to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point1</i> and <i>l_point2</i>	The lower left and upper right coordinates of the bounding box enclosing the arc and the ellipse containing the arc.
<i>n_startAngle</i>	The starting angle, specified in degrees, measured such that 0 degrees is the 3 o'clock position.
<i>n_sweepAngle</i>	Measured in degrees, from the starting angle that the arc will span. A positive degree sweeps the arc in an counterclockwise direction. A negative <i>sweepAngle</i> sweeps the arc in a clockwise direction.
<i>s_tagSymbol</i>	Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddBox

```
dlAddBox(  
    w_dlist  
    x_penNumber  
    l_point1  
    l_point2  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a box to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point1</i> and <i>l_point2</i>	Any two opposite corners of the box.
<i>s_tagSymbol</i>	Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddCircle

```
dlAddCircle(  
    w_dlist  
    x_penNumber  
    l_point  
    x_radius  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a circle to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point</i>	Center point of the circle.
<i>x_radius</i>	Radius of the circle.
<i>s_tagSymbol</i>	Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddDonut

```
dlAddDonut(  
    w_dlist  
    x_penNumber  
    l_point  
    x_innerRadius  
    x_outerRadius  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a donut to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point</i>	Center of the donut.
<i>x_innerRadius</i>	Radius of the inner circle of the donut.
<i>x_outerRadius</i>	Radius of the outer circle of the donut.
<i>s_tagSymbol</i>	Symbol to tag this object with.

dlAddEventObject

```
dlAddEventObject(  
    w_dlist  
    l_point1  
    l_point2  
    t_procName  
    t_procArgs  
    [t_highlightMode]  
    [g_doesGraphics]  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds an *EventObject* to the display list. *EventObjects* are SKILL procedures called whenever the user presses a mouse button or a keyboard key. If the *s_doesGraphics* argument is *t*, the *EventObject* is called whenever the display list is drawn.

Arguments

w_dlist Display list to add to.

l_point1 and *l_point2*

The lower-left and upper-right coordinates of a box, within which the *SkillObject* will draw. The display list processing functions use this box to determine the visibility of the *SkillObject*. The procedure you write to define the *SkillObject* must be written to draw and then return as quickly as possible. If not, the performance of the system will suffer.

t_procName SKILL function to call.

The *t_procName* procedure is called in the following manner:

```
t_procName( l_point1 l_point2 s_eventType  
            t_eventDetail t_procArgs)
```

s_eventType is one of the following symbols:

draw (the display list is drawing)
button1 (the left mouse button was pressed)
button2 (the middle mouse button was pressed)

Cadence User Interface SKILL Functions Reference

Display Lists

`button3` (the right mouse button was pressed)
`key` (a key was pressed)

In the case of the `key` `eventType`, the `t_eventDetail` tells you which key was pressed. Otherwise, the `eventDetail` argument is `nil`.

`t_procArgs` Arguments passed to the function.

`t_highlightMode` When the user enters into an event object, you can specify the system to draw a box around the sensitive area, or invert the sensitive area, or to do nothing. These are specified by `box` (or `b`), `invert` (or `i`), or `none`. The default value is `none`.

`g_doesGraphics` Tells the display list processing functions whether you use `dlDraw` functions inside your SKILL procedure or not. `dlDraw` functions are discussed later on. If you don't call `dlDraw` functions, the rendering of your display list will be faster. This argument should be `t` or `nil`.

`s_tagSymbol` Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddPath

```
dlAddPath(  
    w_dlist  
    x_penNumber  
    x_pathWidth  
    t_pathEndType  
    l_pointsList  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a path to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>x_pathWidth</i>	Width of the path.
<i>t_pathEndType</i>	End type can have one of three values: <code>Truncate</code> , <code>Extend</code> , or <code>Round</code> . <code>Truncate</code> means the path will be square-ended and stop at the two endpoints. <code>Extend</code> means the path will be square-ended, but it will extend 1/2 the width from each end point. <code>Round</code> means the path will have a rounded endpoint and stop at the two endpoints.
<i>l_pointsList</i>	Coordinates of the center line of the path.
<i>s_tagSymbol</i>	Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddPoint

```
dlAddPoint(  
    w_dlist  
    x_penNumber  
    l_point  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a point to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point</i>	List containing the point. For example, list(200 200) or 200:200
<i>s_tagSymbol</i>	Symbol to tag this object with.

Value Returned

It returns `t` on success and `nil` on failure.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddPolygon

```
dlAddPolygon(  
    w_dlist  
    x_penNumber  
    l_pointsList  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a polygon to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_pointsList</i>	<p>Polygon which is a list of points comprising its vertices. For example, the list of points can be specified in either of these ways:</p> <pre>'((0 0) (100 180) (200 200)) list(0:0 100:180 200:200)</pre> <p>You do not need to close the polygon.</p>
<i>s_tagSymbol</i>	Symbol to tag this object with.

dlAddRasterText

```
dlAddRasterText(
    w_dlist
    x_penNumber
    l_point
    t_text
    t_fontName
    [t_verticalJustify]
    [t_horizontalJustify]
    [s_tagSymbol]
)
=> t / nil
```

Description

Adds a text string to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point</i>	Origin point of the text.
<i>t_text</i>	Text string to be added to the display list
<i>t_fontName</i>	X font used to draw text. The font name can vary from system to system. Check your <code>/usr/lib/X11/fonts</code> directory see what fonts are available, or ask your system administrator. Usually fonts <code>fixed</code> and <code>"9x15"</code> are available. If the font name you specify is not found, then <code>fixed</code> is used. Raster text does not scale, but has a fixed width and height.
<i>t_verticalJustify</i>	<p>Presentation of the text, and one of the following strings:</p> <p><code>"dlcTop"</code> (the top of the text will be at the y of the point) <code>"dlcMiddle"</code> (the center of the text will be at the y of the point) <code>"dlcBottom"</code> (the bottom of the text will be at the y of the point)</p> <p>If not specified, <i>t_verticalJustify</i> is set to <code>dlcBottom</code>.</p>

Cadence User Interface SKILL Functions Reference

Display Lists

t_horizontalJustify

Presentation of the text, and one of the following strings:

"dlcLeft" (the string will start at the x of the point)

"dlcCenter" (the middle of the string will be at the x of the point)

"dlcRight" (the end of the string will be at the x of the point)

If not specified, *t_horizontalJustify* is set to dlcLeft.

s_tagSymbol

Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddSegment

```
dlAddSegment(  
    w_dlist  
    x_penNumber  
    l_point1  
    l_point2  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a line segment to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point1</i>	First point of the segment.
<i>l_point2</i>	Second point of the segment.
<i>s_tagSymbol</i>	Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddSkillObject

```
dlAddSkillObject(  
    w_dlist  
    x_penNumber  
    l_point1  
    l_point2  
    t_procName  
    t_procArgs  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a *SkillObject* to the display list. *SkillObjects* are SKILL procedures called whenever the object should be drawn.

Arguments

w_dlist Display list to add to.

x_penNumber Pen number to use.

l_point1 and *l_point2*

The lower-left and upper-right coordinates of a box within which the *SkillObject* will draw. The display list processing functions use this box to determine the visibility of the *SkillObject*. The procedure you write to define the *SkillObject* must be written to draw and then return as quickly as possible. If not, the performance of the system will suffer.

t_procName SKILL function to call.

t_procArgs Arguments passed to the function. The display list processing routines call your function in the following way:

```
t_procName( l_point1 l_point2 t_procArgs )
```

s_tagSymbol Symbol to tag this object with.

Cadence User Interface SKILL Functions Reference

Display Lists

dlAddStrokeText

```
dlAddStrokeText(  
    w_dlist  
    x_penNumber  
    l_point  
    t_text  
    t_justification  
    t_fontName  
    x_height  
    [t_orientation]  
    [g_drafting]  
    [s_tagSymbol]  
)  
=> t / nil
```

Description

Adds a stroke text string to the display list.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to use.
<i>l_point</i>	Origin point of the text
<i>t_text</i>	Text string to be added to the display list
<i>t_justification</i>	One of the following strings: "upperLeft," "centerLeft," "left," "lowerLeft," "upperCenter," "center," "lowerCenter," "upperRight," "centerRight," "right," "lowerRight"
<i>t_fontName</i>	One of the following font names: "euro," "gothic," "math," "roman," "script," "stick," or "swedish"
<i>x_height</i>	Height of text.
<i>t_orientation</i>	One of the following values: "0," "90," "180," "270," "mx," "sideways," "my," "upsideDown," "mr90," "sideways," "mr270," "sideways&270". Default value is "0".

Cadence User Interface SKILL Functions Reference

Display Lists

<i>g_drafting</i>	If this is set to <code>t</code> , the label will always be readable from left to right, or top to bottom. It will never appear mirrored or backward. The default is <code>nil</code> .
<i>s_tagSymbol</i>	Symbol to tag this object with.

Display List Draw Functions

The following `dlDraw` functions are only used inside a *SkillObject* procedure. They render shapes to the screen. The display list processing functions transform the coordinates of your *SkillObject* to the box you specified when you added it to the display list. Therefore, the coordinates you specify when calling a `dlDraw` function are specified such that 0:0 is coincident with the lower-left corner of the SKILL object.

For example, if you said

```
dlAddSkillObject( dlist 4 100:100 200:200
                  "mySkillObject" "10")
dlAddSkillObject( dlist 4 300:300 400:400
                  "mySkillObject" "20")
```

The display list functions call your procedure like so

```
mySkillObject( 100:100 200:200 10)
mySkillObject( 300:300 400:400 20)
```

If you want to draw a circle and have the radius passed to you, define your *SkillObject* as:

```
procedure( mySkillObject( ll ur radius)
  prog( ( )
  dlDrawCircle( 50:50 radius)
  return(t)
  ) ; end prog
  ) ; end procedure
```

There is nothing special about a *SkillObject* procedure, except the first two arguments are always the same, and you are allowed to call `dlDraw` functions. So, the *SkillObject* procedure can do practically anything you want. However, they must do it quickly and never call a function that causes evaluation to block.

There is a `dlDraw` function for all shapes the `dlAddObject` functions provide (except for *dlAddSkillObject*). The arguments to the *dlDraw* functions are the same as the `dlAddObject` functions, except for the *w_dlist* and *x_penNumber* arguments (which are not necessary, since the values from the SKILL object are used).

The `dlDraw` functions all return `t` when the object is drawn, and `nil` if the `dlDraw` function is called outside a *SkillObject* procedure.

Cadence User Interface SKILL Functions Reference

Display Lists

dlDrawArc

```
dlDrawArc(  
    l_point1  
    l_point2  
    f_startAngle  
    f_sweepAngle  
)  
=> t / nil
```

Description

Draws an arc to the screen from within a *SkillObject* procedure.

Arguments

<i>l_point1</i> and <i>l_point2</i>	The lower-left and upper-right corners of a box enclosing the arc.
<i>f_startAngle</i>	Measured such that 0 degrees is the 3 o'clock position.
<i>f_sweepAngle</i>	Measured in degrees, from the starting angle the arc will span. A positive degree sweeps the arc in an counterclockwise direction. A negative <i>sweepAngle</i> sweeps the arc in a clockwise direction.

dlDrawBox

```
dlDrawBox(  
    l_point1  
    l_point2  
)  
=> t / nil
```

Description

Draws a box to the screen from within a *SkillObject* procedure.

Arguments

l_point1 and *l_point2*
Any two opposite corners of a box.

dlDrawCircle

```
dlDrawCircle(  
    l_point  
    x_radius  
)  
=> t / nil
```

Description

Draws a circle to the screen from within a *SkillObject* procedure.

Arguments

<i>l_point</i>	Center point of circle.
<i>x_radius</i>	Radius of the circle.

Cadence User Interface SKILL Functions Reference

Display Lists

dlDrawDonut

```
dlDrawDonut(  
    l_point  
    x_innerRadius  
    x_outerRadius  
)  
=> t / nil
```

Description

Draws a donut to the screen from within a *SkillObject* procedure.

Arguments

<i>l_point</i>	Center point of donut.
<i>x_innerRadius</i>	Inside radius of donut.
<i>x_outerRadius</i>	Outside radius of donut.

Cadence User Interface SKILL Functions Reference

Display Lists

dlDrawPath

```
dlDrawPath(  
    x_pathWidth  
    t_pathEndType  
    l_pointsList  
)  
=> t / nil
```

Description

Draws a path to the screen from within a *SkillObject* procedure. A path is specified by a center line and a width.

Arguments

<i>x_pathWidth</i>	Path width.
<i>t_pathEndType</i>	End type can have one of three values: <i>Truncate</i> , <i>Extend</i> or <i>Round</i> . <i>Truncate</i> means the path will be square-ended and stop at the two endpoints. <i>Extend</i> means the path will be square-ended, but it will extend 1/2 the width from each end point. <i>Round</i> means the path will have a rounded endpoint and stop at the two endpoints.
<i>l_pointsList</i>	Coordinates of the center line.

Cadence User Interface SKILL Functions Reference

Display Lists

dlDrawPoint

```
dlDrawPoint(  
    l_point  
)  
=> t / nil
```

Description

Draws a point to the screen from within a *SkillObject* procedure.

Arguments

<i>l_point</i>	Position of the point.
----------------	------------------------

dlDrawPolygon

```
dlDrawPolygon(  
    l_pointsList  
)  
=> t / nil
```

Description

Draws a polygon to the screen from within a *SkillObject* procedure.

Arguments

<i>l_pointsList</i>	List of points comprising its vertices. You do not need to close the polygon.
---------------------	---

dlDrawRasterText

```
dlDrawRasterText(  
    l_point  
    t_text  
    t_fontName  
    [t_verticalJustify]  
    [t_horizontalJustify]  
)  
=> t / nil
```

Description

Draws a text string to the screen from within a *SkillObject*. The text is drawn using an X Window System font.

Arguments

<i>l_point</i>	Origin point of the text.
<i>t_text</i>	Text string to be added to the display list.
<i>t_fontName</i>	X font used to draw text. The font name can vary from system to system. Check your <code>/usr/lib/X11/fonts</code> directory see what fonts are available, or ask your system administrator. Usually fonts "fixed" and "9x15" are available. If the font name you specify is not found, then "fixed" is used. Raster text does not scale, but has a fixed width and height.
<i>t_verticalJustify</i>	<p>Presentation of the text, and one of the following strings:</p> <p>"dlcTop" (the top of the text will be at the y of the point) "dlcCenter" (the center of the text will be at the y of the point) "dlcBottom" (the bottom of the text will be at the y point)</p> <p>If not specified, <i>t_verticalJustify</i> is set to "dlcBottom".</p>
<i>t_horizontalJustify</i>	<p>Presentation of the text, and one of the following strings:</p> <p>"dlcLeft" (the string will start at the x of the point) "dlcCenter" (the middle of the string will be at the x of the</p>

Cadence User Interface SKILL Functions Reference

Display Lists

point)

"dlcRight" (the end of the string will be at the x of the point)

If not specified, *t_horizontalJustify* is set to
"dlcLeft".

dlDrawSegment

```
dlDrawSegment(  
    l_point1  
    l_point2  
)  
=> t / nil
```

Description

Draws a line segment to the screen from within a *SkillObject* procedure.

Arguments

<i>l_point1</i>	First point of the segment, assuming 0:0 to be the lower-left corner.
<i>l_point2</i>	Second point of the segment, assuming 0:0 to be the lower-left corner.

dlDrawStrokeText

```
dlDrawStrokeText(  
    l_point  
    t_text  
    t_justification  
    t_fontName  
    x_height  
    t_orientation  
)  
=> t / nil
```

Description

Draws a stroke text string to the screen from within a *SkillObject* procedure. The height (and therefore scale) can be specified.

Arguments

<i>l_point</i>	Origin point of the text.
<i>t_text</i>	Text string to be added to the display list.
<i>t_justification</i>	One of the following strings: "upperLeft," "centerLeft," "left," "lowerLeft," "upperCenter," "center," "lowerCenter," "upperRight," "centerRight," "right," "lowerRight"
<i>t_fontName</i>	One of the following font names: "euro," "gothic," "math," "roman," "script," "stick," "small," "large," "mask," "swedish"
<i>x_height</i>	Height of text.
<i>t_orientation</i>	One of the following values: "0," "90," "180," "270," "mx," "sideways," "my," "upsideDown," "mr90," "sideways," "mr270," "sideways&270"

Display List Pens

Display list pens store the graphic attributes of an object. These attributes are

- Color
- FillStyle
- FillPattern
- LineStyle
- LinePattern

Pens are referred to by number. Pens are also grouped together into pen tables. In the default pen table, there are 64 pens and they are numbered 0 through 63.

When you create a display list, it is initialized to use a default pen table. Changes made to a pen table affect all the display lists that use that pen table, much like changing a technology file changes all the cells that use it. To allow you to have a unique pen table, functions are provided to create new pen tables and to specify that a display list should use a particular pen table.

The color of the pens is set by color index. As such, the display list pens do not allocate any colors, but share the colors already in the color table. Therefore, they must be considered read-only. In order to set a pen's color, you must call [hiMatchColor](#) to find the color index that most closely matches the color you desire.

Fill patterns and Line patterns are different; you can create and change these patterns at will.

Pens also specify the order in which objects are rendered. When display lists are drawn, all the objects on pen 0 are drawn, then all the objects on pen 1 are drawn, and so on up to pen 63. As such, objects on pen 4 draw on top of objects on pen 3.

Cadence User Interface SKILL Functions Reference

Display Lists

dlMakePenTable

```
dlMakePenTable(  
    [x_maxPens]  
)  
=> w_penTable / nil
```

Description

Creates a pen table a display list can use.

Arguments

x_maxPens

Number of pens in the pen table. It is best to create only as many pens as you plan to use; any unused pens waste memory. The pen numbers range from 0 to *x_maxPens*-1. The function returns the pen table, or `nil` if an error occurred. If not specified, the default number of pens is 64.

Cadence User Interface SKILL Functions Reference

Display Lists

dlMakeStipple

```
dlMakeStipple(  
    t_StippleString  
)  
=> x_stippleId / nil
```

Description

Creates a stipple pattern from the supplied string.

Arguments

t_StippleString String that has a special form. The string is composed of 1s, 0s and newline characters `\n` and " " (space) characters. The string describes an array of bits that define a stipple pattern. Each newline `\n` or " " (space) character in the string begins a new row of bits. Each row must have the same number of characters in it, or the functions return `nil`. This is a sample stipple pattern:

```
stipString = "0101\n\  
1010\n\  
0101\n\  
1010"
```

```
stippleId = dlMakeStipple( stipString)
```

In this example, a 4x4 stipple pattern is created and its ID returned. The 1 character specifies the bits turned on, and the 0 character returns the bits turned off. The maximum size of a pattern is 64x64.

Value Returned

It returns the *stippleId*, or `nil` if an error occurred.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetCurrentPen

```
dlSetCurrentPen(  
    x_penNumber  
)  
=> t / nil
```

Description

Sets the specified pen number to the current pen. Changes the pen number objects are drawn with from within a *SkillObject* procedure. This function should only be called from within a *SkillObject* procedure.

Arguments

<i>x_penNumber</i>	Pen number to set.
--------------------	--------------------

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetPenColor

```
dlSetPenColor(  
    x_penNumber  
    x_colorIndex  
    [w_penTable]  
)  
=> t / nil
```

Description

Changes the specified pen to the specified color index from the pen table.

Arguments

<i>x_penNumber</i>	Pen number to use.
<i>x_colorIndex</i>	Color desired.
<i>w_penTable</i>	Pen table this pen belongs to. If you do not specify the <i>w_penTable</i> , the default pen table is used. In order to obtain a desired color, you may use <u>hiMatchColor</u> to find the color index that most closely matches the desired color.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetPenFillStyle

```
dlSetPenFillStyle(  
    x_penNumber  
    t_fillStyle  
    [w_penTable]  
)  
=> t / nil
```

Description

Sets the fill style of a pen.

Arguments

x_penNumber Pen number to be set.

t_fillStyle One of the following strings:

"OutlineNoFill"	the default condition
"SolidFill"	all objects are filled
"XFill"	same as "OutlineNoFill" except boxes have an X drawn inside
"StippleFill"	objects are filled with the stipple pattern
"OutlineStippleFill"	objects are drawn stipple filled with a solid line drawn around the outside

w_penTable Pen table to use. If not specified, the default pen table is used.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetPenFilled

```
dlSetPenFilled(  
    x_penNumber  
    [w_penTable]  
)  
=> t / nil
```

Description

Sets the fill style of a pen to `SolidFill`.

Arguments

<i>x_penNumber</i>	Pen number to be set.
<i>w_penTable</i>	Pen table to use. If you do not specify this argument, the default pen table is used.

Value Returned

t	The fill style was set.
nil	The fill style could not be set.

dlSetPenStipple

```
dlSetPenStipple(  
    x_penNumber  
    x_stippleId  
    [w_penTable]  
)  
=> t / nil
```

Description

Sets the stipple pattern for a pen.

Arguments

<i>x_penNumber</i>	Pen number to be set.
<i>x_stippleId</i>	Stipple pattern used.
<i>w_penTable</i>	Pen table to set. If you do not supply <i>w_penTable</i> , then the pen in the default pen table will be set, otherwise the pen in the pen table specified by will be set.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetPenTable

```
dlSetPenTable(  
    w_displayList  
    w_penTableOrDisplayList  
)  
=> t / nil
```

Description

Sets the pen table a display list will use.

Arguments

w_displayList Display list you wish to set.

w_penTableOrDisplayList
Either a pen table or another display list. If the argument is a display list, the *w_displayList* is set to use the same pen table as the second *displayList*.

Viewing a Display List

A display list can be viewed in empty windows. A single display list can be attached to as many windows as you wish. They are scaled to fit within a window.

dlAttachDlistToWidget

```
dlAttachDlistToWidget(  
    w_displayList  
    x_widgetId  
)  
=> x_widgetId / nil
```

Description

Attaches a display list to a widget.

Arguments

<i>w_displayList</i>	Display list to attach.
<i>x_widgetId</i>	Widget to attach the display list to.

dlAttachDlistToWindow

```
dlAttachDlistToWindow(  
    w_displayList  
    w_windowId  
)  
=> t / nil
```

Description

Attaches a display list to the window defined by *w_windowId*. This function can be used to preview a display list.

Arguments

<i>w_displayList</i>	Display list you wish to attach to the window.
<i>w_windowId</i>	ID number of an empty window. It cannot be the window of another application.

Cadence User Interface SKILL Functions Reference

Display Lists

dlClearDisplayList

```
dlClearDisplayList(  
    w_displayList  
)  
=> t / nil
```

Description

Removes all the objects from a display list.

Arguments

w_displayList Display list you are clearing of all objects.

Value Returned

It returns `t` if it could remove all the objects, or `nil` if an error occurred.

Cadence User Interface SKILL Functions Reference

Display Lists

dlCloseWidget

```
dlCloseWidget(  
    x_widgetId  
)  
=> t / nil
```

Description

Closes (destroys) a widget.

Arguments

<i>x_widgetId</i>	Widget to destroy.
-------------------	--------------------

dlConfigureButton

```
dlConfigureButton(  
    x_button  
    x_xPosition  
    x_yPosition  
    x_width  
    x_height  
)  
=> t / nil
```

Description

Moves a button to another location. Both the x and y position are specified in X Window System coordinates, where 0:0 is the upper left corner of the screen.

Arguments

<i>x_button</i>	Button being moved.
<i>x_xPosition</i>	New x position of the upper-left corner of the button.
<i>x_yPosition</i>	New y position of the upper-left corner of the button.
<i>x_width</i>	Width of new button size.
<i>x_height</i>	Height of new button size.

dlDetachDlistFromWidget

```
dlDetachDlistFromWidget(  
    w_displayList  
    x_widgetId  
)  
=> t / nil
```

Description

Detaches a display list from a widget.

Arguments

<i>w_displayList</i>	Display list to detach.
<i>x_widgetId</i>	Widget to detach the display list from.

Cadence User Interface SKILL Functions Reference

Display Lists

dlDetachDlistFromWindow

```
dlDetachDlistFromWindow(  
    w_displayList  
    w_windowId  
)  
=> t / nil
```

Description

Detaches a display list (*w_displayList*) from the window defined by *w_windowId*.

Arguments

<i>w_displayList</i>	Display list you wish to detach from the window.
<i>w_windowId</i>	ID number of an empty window. It cannot be the window of another application.

Cadence User Interface SKILL Functions Reference

Display Lists

dlDisplay

```
dlDisplay(  
    w_displayList  
)  
=> t / nil
```

Description

Redraws the display list (*w_displayList*) in all the windows and buttons it is currently attached to.

Arguments

<i>w_displayList</i>	Display list to redraw.
----------------------	-------------------------

Cadence User Interface SKILL Functions Reference

Display Lists

dlDlistToIcon

```
dlDlistToIcon(  
    w_displayList  
    x_width  
    x_height  
    [x_backColorIndex]  
)  
=> l_iconId / nil
```

Description

Converts a displayList to an icon.

Arguments

<i>w_displayList</i>	Display list.
<i>x_width</i>	Width of icon specified in screen units.
<i>x_height</i>	Height of icon specified in screen units.
<i>x_backColorIndex</i>	Background color of the generated icon. If you do not specify this argument, the icon will have the same background color as Graphic Editor windows. The returned <i>l_iconId</i> can be used in forms or as the window manager icon for a window or windows.

Cadence User Interface SKILL Functions Reference

Display Lists

dlEnableItem

```
dlEnableItem(  
    w_dlist  
    x_penNumber  
    s_symbolTag  
    g_enableOrDisable  
)  
=> t / nil
```

Description

Allows you to turn on or off shapes within a display list that match the provided symbol. All shapes on pen *x_penNumber* are searched and checked to see if the *symbolTag* matches.

Arguments

<i>w_dlist</i>	Display list to add to.
<i>x_penNumber</i>	Pen number to turn shapes on or off.
<i>s_symbolTag</i>	Symbol to match to all shapes on pen <i>x_penNumber</i> .
<i>g_enableOrDisable</i>	<i>t</i> enables the shape; <i>nil</i> disables the shape.

Cadence User Interface SKILL Functions Reference

Display Lists

dIFitDlistOnDraw

```
dIFitDlistOnDraw(  
    w_displayList  
    x_widgetId  
    g_doFit  
)  
=> t / nil
```

Description

Specifies whether a display list should automatically fit within the specified widget.

Arguments

<i>w_displayList</i>	Display list.
<i>x_widgetId</i>	ID the display list is attached to.
<i>g_doFit</i>	(<i>t</i> or <i>nil</i>) <i>t</i> means the display list should be automatically fitted in the widget; <i>nil</i> means it should not.

dlMakeDlistButton

```
dlMakeDlistButton(  
    x_parentWidgetId  
    x_xPosition  
    x_yPosition  
    x_width  
    x_height  
    w_displayList  
    t_pressCallback  
    [x_backgroundColor]  
)  
=> t / nil
```

Description

Creates a button to display a display list in. The display list button is a rectangular region with a widget or window in which a display list is drawn and which dispatches a callback whenever the left mouse button is pressed within it.

Arguments

<i>x_parentWidgetId</i>	Parent widget to make the button within.
<i>x_xPosition</i>	x position of the upper-left corner of the button, in X Window System screen coordinates (where 0:0 is the upper-left corner of the widget).
<i>x_yPosition</i>	y position of the upper-left corner of the button, in X Window System screen coordinates (where 0:0 is the upper-left corner of the widget).
<i>x_width</i>	Width of the button.
<i>x_height</i>	Height of the button.
<i>w_displayList</i>	Display list to attach to the button.
<i>t_pressCallback</i>	SKILL expression evaluated whenever the left mouse button is pressed.
<i>x_backgroundColor</i>	Optional background color of the button.

Cadence User Interface SKILL Functions Reference

Display Lists

dlMakeWidget

```
dlMakeWidget(  
    [x_backgroundColor]  
    [l_point1]  
    [l_point2]  
)  
=> x_widgetId / nil
```

Description

Creates a general purpose window (or widget). Uses of this widget can be for fixed menus or graphic display panels. Both points are specified in X Window System coordinates, where 0:0 is the upper-left corner of the screen.

Arguments

<i>x_backgroundColor</i>	Background color of the widget.
<i>l_point1</i>	Upper left position of the widget.
<i>l_point2</i>	Lower right position of the widget.

Cadence User Interface SKILL Functions Reference

Display Lists

dlMapWidget

```
dlMapWidget(  
    x_widgetId  
)  
=> t / nil
```

Description

Puts a widget back on the screen after a call to `dlUnMapWidget`. This function also applies to buttons.

Arguments

<i>x_widgetId</i>	Widget to map.
-------------------	----------------

Cadence User Interface SKILL Functions Reference

Display Lists

dlMoveButton

```
dlMoveButton(  
    x_button  
    x_xPosition  
    x_yPosition  
)  
=> t / nil
```

Description

Moves a button to another location. Both the x and y position are specified in X Window System coordinates, where 0:0 is the upper-left corner of the screen.

Arguments

<i>x_button</i>	Button being moved.
<i>x_xPosition</i>	New x position of the upper-left corner of the button.
<i>x_yPosition</i>	New y position of the upper-left corner of the button.

dlResizeButton

```
dlResizeButton(  
    x_button  
    x_width  
    x_height  
)  
=> t / nil
```

Description

Resizes a button.

Arguments

<i>x_button</i>	Button being moved.
<i>x_width</i>	Width of new button size.
<i>x_height</i>	Height of new button size.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSaveDlist

```
dlSaveDlist(  
    w_displayList  
    t_fileName  
    t_dlistName  
)  
=> t / nil
```

Description

Saves the display list.

Arguments

<i>w_displayList</i>	Display list.
<i>t_fileName</i>	Name of file whose format is the SKILL functions necessary to regenerate the display list. When loaded, all the add functions will be called, and a display list will be created.
<i>t_dlistName</i>	Name of the display list.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetDlistPosition

```
dlSetDlistPosition(  
    w_displayList  
    x_widgetId  
    x_xOffset  
    x_yOffset  
)  
=> t / nil
```

Description

Specifies the scale factor to map from display list coordinates to screen coordinates. The offsets only have effect if auto-fitting was turned off (via `dfIIFitDlistOnDraw`).

Arguments

<i>w_displayList</i>	Display list.
<i>x_widgetId</i>	ID the display list is attached to.
<i>x_xOffset</i>	Offset (in display list coordinates) of the display list in the x axis, assuming 0:0 to be the upper-left corner.
<i>x_yOffset</i>	Offset (in display list coordinates) of the display list in the y axis, assuming 0:0 to be the upper-left corner.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetDlistScale

```
dlSetDlistScale(  
    w_displayList  
    x_widgetId  
    f_scale  
)  
=> t / nil
```

Description

Specifies the scale factor to map from display list coordinates to screen coordinates. This scale factor only has effect if auto-fitting was turned off (using `dfIIFitDlistOnDraw`).

Arguments

<i>w_displayList</i>	Display list.
<i>x_widgetId</i>	ID the display list is attached to.
<i>f_scale</i>	Scale factor.

Cadence User Interface SKILL Functions Reference

Display Lists

dlSetWidgetName

```
dlSetWidgetName(  
    x_widgetId  
    t_name  
)  
=> t / nil
```

Description

Sets the text displayed by the window manager banner.

Arguments

<i>x_widgetId</i>	Widget to name.
<i>t_name</i>	Text to name the widget.

Cadence User Interface SKILL Functions Reference

Display Lists

dlUnMapWidget

```
dlUnMapWidget(  
    x_widgetId  
)  
=> t / nil
```

Description

Removes the widget from the screen, but does not destroy it. This function also applies to buttons.

Arguments

<i>x_widgetId</i>	Widget to unmap.
-------------------	------------------

A Sample Display List

The following SKILL program makes a display list to draw inside a window and to use as its icon.

```
/* First, make the DisplayList. */
dl = dlMakeDisplayList( )
/* Now make a pen table for it. */
penTable = dlMakePenTable(5)
/* Assign the penTable to the display list. */
dlSetPenTable( dl penTable)
/* Define a couple of colors. */
colorIndex = hiMatchColor( nameToColor( "blue"))
dlSetPenColor( 1 colorIndex penTable)
/* Set this pen "filled." */
dlSetPenFillStyle( 1 "SolidFill" penTable)
/* Put the objects in. */
dlAddBox( dl 1 0:0 100:100)
dlAddBox( dl 2 10:10 90:90)
dlAddCircle( dl 3 50:50 20)
dlAddStrokeText( dl 4 50:50 "Window" "0" "gothic" 25 "center")
/* Make a window. Put the DisplayList into the window */
/* and set the window's icon. */
w = hiOpenWindow( )
dlAttachDlistToWindow( dl w)
/* Convert the dlist to an icon. */
icon = dlDlistToIcon( dl 50 50)
hiSetWindowIcon( w icon)
/* Save it to a file. */
dlSaveDlist( dl "save.dlist" "newDl")
/* Try resizing the window. The dlist will fit inside it. */
/* Iconify the window. */
```

Cadence User Interface SKILL Functions Reference
Display Lists

Graph Browser

The following topic is discussed in this appendix:

- [Overview](#) on page 762
- [Key Terms](#) on page 764
- [Properties of the dagArc](#) on page 766
- [Properties of the dagClass](#) on page 767
- [Properties of the dagNode](#) on page 769
- [Properties of the dagTool](#) on page 772
- [Actions and Action Lists](#) on page 777
- [dag Functions](#) on page 779

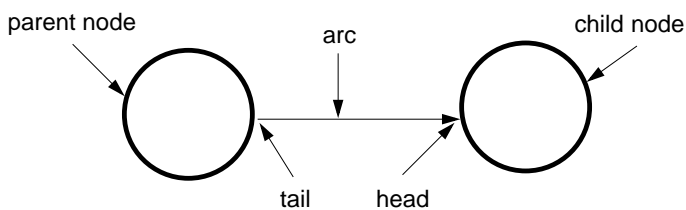
Overview

This section discusses the Directed Acyclic Graph (*dag*) interface, which you can use to create a browser tool. A browser tool enables a user to interactively display and construct or modify directed graphs.

The section contains

- An introduction of key terms
- Tables describing the properties of dag object types
- A description of action lists
- Dag browser SKILL functions in alphabetical order

A *dag* is a generic data structure that is commonly used in computer programs to represent objects and show how those objects relate to each other. The objects are represented in a directed graph by nodes. How the nodes relate to each other is represented by arcs that connect the nodes together. The graph is classified as *directed* because the arc connecting any two nodes together has a direction. The direction of the arc is from the parent node (at the tail of the arc) to the child node (at the head of the arc).



Although the name *dag* implies that the graph cannot have cycles, you can make cycles in the graphs you construct with the dag interface. When the dag functions traverse the graph you make, they take the necessary precautions to avoid infinitely looping because of cycles in the graph. However, the automatic placement or routing of graphs with cycles might not be visually pleasing.

The Design Framework II dag interface provides many sophisticated capabilities for constructing browser tools. These include

- The concept of classes of nodes, with inheritance of information from class to node.
- Context-sensitive pop-up menus and bind keys, based on the class of the node over which the menu is invoked.
- Many capabilities for controlling the layout and appearance of the graph presentation.

Cadence User Interface SKILL Functions Reference

Graph Browser

- Automatic placement and routing algorithms for the nodes and arcs in the graph.
- Ability to display a text string and an icon for each node. The icon can be customized using the display list interface. The icon and text strings displayed can be node-specific or class-specific.
- Ability to use any number of browser tools simultaneously.
- Ability to share nodes, arcs, and classes among many tools. The same node can have different appearances in different tools.
- Ability to use many of the normal *hi* interface functions for a window, including pulldown menus for a dag browser window.

Key Terms

Coordinate Systems

There are two coordinate systems used in specifying dag object properties and dag function parameters:

- *screen coordinates*. These have their origin at the upper left and are specified in terms of pixels.
- *dag coordinates*. These have their origin at the lower left. You determine the magnitude of these coordinates by the height and width you assign to classes.

Context-Sensitive Pop-Up Menus and Actions

You can define context-sensitive pop-up menus and accelerators (bind keys) for nodes or classes. The specifications for these are called action lists. See the action list section.

Display Lists, Icons, and SKILL Objects

You can use the display list interface to create icons for display in the browser. To make those icons reflect information available only at run time, you can use the SKILL object feature of display lists. See the display list chapter. You can also interactively create display lists without SKILL objects using the Graphics Editor *Save Icon* command.

Intrinsic Property Inheritance

Browser nodes inherit a subset of the intrinsic properties of their class. There is a distinction between properties that you add to a browser object (user-defined) and those that are there when an object is created (intrinsic). Only intrinsic properties are inherited.

Inheritance means that, if a node does not have the desired property, the dag functions look for a property by the same name in the node's class. If it does not find the named property in either the node or class, it returns `nil`.

Kinds of Browser (dag) Objects

The tables in the next few pages describe the intrinsic properties of the objects that you can create and manipulate through the dag interface. The object types are: class, node, arc, and

Cadence User Interface SKILL Functions Reference

Graph Browser

tool. Each of these can have user-defined properties in addition to the intrinsic properties listed.

Window Types and Their Properties

A window type is a basic SKILL data type. Dag arc, class, node, and tool objects all are of this data type. In the type specifications for dag properties and dag function parameters and return values, a *w* prefix indicates that the object must be this data type.

The straight arrow notation (*->*) is used to access and modify both the intrinsic and user-defined properties of browser objects. For example, to find out the name of a class object, you would enter

```
name = class->name
```

and, to change the name of a class, you would enter

```
class->name = "newName"
```

Unless otherwise specified all attributes of browser objects can be modified as shown above.

Properties of the dagArc

The table shows the properties of the dagArc. The text that follows explains each property in more detail.

Properties of the dagArc

Name	SKILL Type	Description
type	symbol	Type of object (dagArc).
head	wType (node)	Node attached to head of arc.
tail	wType (node)	Node attached to tail of arc.
arcPen	fixNum	DisplayList Pen number for arc.
arcWidth	fixNum	Width of this arc in a dagTool.

type	Type of the object. The symbol for this object is <i>dagArc</i> . This property is read-only.
head	dagNode that this arc is attached to at its head. This property is read-only. It can be <i>nil</i> , meaning there is no child node attached to the arc.
tail	dagNode that this arc is attached to at its tail. This property is read-only. It can be <i>nil</i> , meaning there is no parent node attached to the arc.
arcPen	Display list pen number that is used to render this arc when it is drawn within a browser tool.
arcWidth	Width of the arc when it is drawn in a browser tool. If the <i>arcWidth</i> is zero, a thin line is drawn; if it is greater than zero, the arc is rendered as a path. The <i>arcWidth</i> dimension is unit-less and is scaled proportionally within the browser tool.

Properties of the dagClass

The table shows the properties of the dagClass. The text that follows explains each property in more detail.

A dagClass is a mechanism for grouping properties that are common to multiple nodes. You specify what the class is for a node when you create the node; each node is in exactly one class. Classes are never directly displayed. A class can only affect what is displayed by the properties inherited by nodes of the class.

Properties of the dagClass

Name	SKILL Type	Description
name	string	Name of the class.
type	symbol	Type of the object (dagClass).
actionList	list	List of actions for this class.
displayList	wType	Display list used to render nodes of this class.
width	fixNum	Width of the nodes of this class.
height	fixNum	Height of the nodes of this class.

name Name of the class.

type Type of the object. The symbol for this object is dagClass. This property is read-only.

The action list, display list, width, and height properties described below are inheritable.

actionList List of actions this class can do. actionLists are described in more detail later in this chapter. This property is read-only. The default value is nil.

displayList Display list nodes of this class used when drawn in a browser tool.

width Width of all the nodes of this class when displayed in a browser tool.

height Height of all the nodes of this class when displayed in a browser tool.

Cadence User Interface SKILL Functions Reference

Graph Browser

The width and height properties are expressed using cardinal numbers. The dimensions are unit-less and can be any range that you wish. The browser tool scales these dimensions, much like the Graphics Editor does when displaying a database.

Properties of the dagNode

The table shows the properties of the dagNode. The text that follows explains each property in more detail.

Properties of the dagNode

Name	SKILL Type	Description
name	string	Name of the node.
textColor	fixNum	Color of the name text when <i>textOnly</i> is <code>t</code> for the tool.
buttonColor	fixNum	Color of this node when displayed when <i>textOnly</i> is <code>t</code> for the tool.
label	string	Text label drawn in the graph (if the dagTool has the <i>showLabels</i> property set to <code>t</code>).
type	symbol	Type of object (dagNode).
actionList	list	List of actions unique to this node.
expandedActions	list	Full list of actions that this node can do, including actions inherited from class.
displayList	wType	Display list used to render the node.
class	wType	Class of this node.
parentObjects	list	Parent nodes of this node.
parents	list	Parent arcs of this node.
childObjects	list	Child nodes of this node.
children	list	Child arcs of this node.
x	fixNum	X position of node when displayed.
y	fixNum	Y position of node when displayed.
invisible	[<code>t</code> or <code>nil</code>]	Node does not display.
pruned	<code>t</code> , <code>nil</code> , or <code>pruneChildren</code>	Node and its children should not be processed or drawn by the dag functions.

Cadence User Interface SKILL Functions Reference

Graph Browser

name	Name of the node. This name is automatically displayed to represent the node in the graph when <i>textOnly</i> is <code>t</code> for the tool. This property is read-only.
textColor	Color that the name of this node is drawn in when displayed in a dagTool. This property only takes effect when <i>textOnly</i> is <code>t</code> for the tool.
buttonColor	Background color that this node will have when displayed in a dagTool when <i>textOnly</i> is <code>t</code> for the tool.
label	The value of this property for the node is shown when <i>showLabels</i> is <code>t</code> for the tool. The label is shown in addition to what would otherwise have been displayed for the node based on the value of <i>textOnly</i> for the tool. The default is the empty string (<code>""</code>).
type	Type of this object. The symbol for this object is <i>dagNode</i> . This property is read-only.
actionList	List of actions that this node can do. These are only the actions that are specific to this node; see <i>expandedActions</i> below. Action lists are described in more detail in the action list section. This property is read-only. The default is <code>nil</code> .
expandedActions	List of all the actions that a node can do. This list is the concatenation of the actions specified on the class and those specified on the node. This property is read-only. The default is <code>nil</code> .
displayList	Display list that this node uses when drawn in a browser tool. It must either be defined for the node or inherited from its class. When <i>textOnly</i> is <code>t</code> for the tool, the contents of the display list property will be ignored.
class	<p>Class of this node (as returned by <i>dagCreateClass</i>). This property is read-only.</p> <p>The <i>dagNode</i> properties <i>parents</i>, <i>parentObjects</i>, <i>children</i>, and <i>childObjects</i> are created whenever a linking function is called to link two nodes together. They are modified whenever an unlinking function is called to break two nodes apart. These</p>

Cadence User Interface SKILL Functions Reference

Graph Browser

	properties can be modified only by calling the linking and unlinking functions; they are read-only properties of the node.
<code>parentObjects</code>	List of nodes that this node attaches as parents.
<code>parents</code>	List of <i>dagArcs</i> that this node attaches as parents.
<code>childObjects</code>	List of nodes that this node attaches as children.
<code>children</code>	List of <i>dagArcs</i> that this node attaches as children.
<code>x</code>	X position in a <i>dagTool</i> at which this node resides. Node positions are specified relative to the lower left corner of the window and of the node bounding box.
<code>y</code>	Y position in a <i>dagTool</i> at which this node resides. Node positions are specified relative to the lower left corner of the window and of the node bounding box.
<code>invisible</code>	Property that (if true) causes the dag functions not to display the node and automatic routing not to route to it. However, automatic placement will reserve room for it in the placed graph.
<code>pruned</code>	Property that (if true) causes the dag functions to not display, reserve space in automatic placement for or route in automatic routing to this node and any of its children. If property is <i>pruneChildren</i> , then this node will display, but its children will not.

Properties of the dagTool

The table shows the properties of the dagTool. The text that follows explains each property in more detail.

A dagTool associates a window with a graph to display in that window. The dagTool is an object (just like nodes and arcs) and has properties that you can reference and assign. These properties govern the display of the graph in this window.

Properties of the dagTool

Name	SKILL Type	Description
name	string	Name of the tool.
type	symbol	Type of object (dagTool).
window	wType	Window of this tool.
toolNumber	fixNum	Number of this tool.
horizontal	[t or nil]	Show left->right.
scaleToFit	[t or nil]	Always fit the graph.
noPlace	[t or nil]	Don't use the placer.
noRoute	[t or nil]	Don't use the router.
preDisplayProc	string	Procedure to call before displaying.
postDisplayProc	string	Procedure to call after displaying.
closeProc	string	Procedure to call when tool is closed.
startingObject	wType	Top node of the tool.
anchorObject	wType	Node to justify to.
anchorTo	symbol	Justification style.
x	fixNum	X position on screen.
y	fixNum	Y position on screen.
width	fixNum	Width of tool.
height	fixNum	Height of tool.
dagLx	fixNum	Lower left x of dag extents.
dagLy	fixNum	Lower left y of dag extents.

Cadence User Interface SKILL Functions Reference

Graph Browser

Properties of the dagTool

Name	SKILL Type	Description
dagUx	fixNum	Upper right x of dag extents.
dagUy	fixNum	Upper right y of dag extents.
viewLx	fixNum	Lower left x of the visible dag.
viewLy	fixNum	Lower left y of the visible dag.
viewUx	fixNum	Upper right x of the visible dag.
viewUy	fixNum	Upper right y of the visible dag.
textOnly	[t or nil]	Show nodes as text (not icons).
showLabels	[t or nil]	t turns on label display mode.
labelJustification	symbol	Where to place label relative to each dag node.
fontName	string	Name of the font to use for labels.

name	Name of the tool. This is displayed in the window banner of the tool window.
type	Type of the object. The symbol for this object is <i>dagClass</i> . This property is read-only.
window	Window of the dagTool. It's initialized when the <i>dagOpenTool</i> function is called. This property is read-only.
toolNumber	Number of this tool. Each tool is assigned a unique number upon creation. This property is read-only.
horizontal	Tells the dagTool to draw the nodes from left to right, rather than from top to bottom. The default is <i>nil</i> , meaning draw the graph from the top to the bottom.
scaleToFit	Tells the <i>dagTool</i> to draw the graph so that bounding boxes of all of its nodes fit within the given window. Icons and scalable font text strings are scaled proportionally; raster text strings are not. If <i>nil</i> , the dagTool uses the view box (specified by <i>viewLx</i> , <i>viewLy</i> , <i>viewUx</i> , and <i>viewUy</i>) and draws only the visible, unpruned nodes that exist within this box. The default is <i>t</i> . Using

Cadence User Interface SKILL Functions Reference

Graph Browser

	this property causes the <i>anchorTo</i> and <i>anchorObject</i> properties to be ignored.
<code>noPlace</code>	Tells the <i>dagTool</i> not to invoke the placer. If you set this to <code>t</code> , then you are responsible for positioning the nodes yourself. The default is <code>nil</code> .
<code>noRoute</code>	If <code>t</code> tells the <i>dagTool</i> not to invoke the router. The default is <code>nil</code> .
	The pre and post display procedures can be used to do work just before or just after the graph is displayed.
<code>preDisplayProc</code>	<p>The <code>preDisplayProc</code> is called with the following form:</p> <pre>preDisplayProc(tool dirtyOrNot)</pre> <p>The <i>dirtyOrNot</i> parameter is true if any nodes in the graph have been modified (either by having new children created, or by having a property modified).</p> <p>If you have disabled automatic placement, you can do the placement you want in this function.</p>
<code>postDisplayProc</code>	<p>The <i>postDisplayProc</i> is called with the following form:</p> <pre>postDisplayProc(tool)</pre>
<code>closeProc</code>	<p>SKILL procedure that is called when the <i>dagTool</i> is closed. The <i>closeProc</i> takes the following arguments.</p> <pre>closeProc(w_dagTool)</pre>
<code>startingObject</code>	<i>dagNode</i> that the tool starts with. This node is the root of the graph you want displayed in the tool. This property is read-only.
<code>anchorObject</code>	Node the tool justifies to. The <i>dagTool</i> positions its viewing box about this node. See the <i>anchorTo</i> property.
<code>anchorTo</code>	Justification style for positioning the graph around the <i>anchorObject</i> . The graph is justified around the corresponding part of the <i>anchorObject</i> node. It is ignored when the <i>scaleToFit</i>

Cadence User Interface SKILL Functions Reference

Graph Browser

property is `t`. It can be one of the symbols name on the left in the following table or *'none*.

center	Center the object in the tool.
centerLeft	Vertical center, horizontal to left.
centerRight	Vertical center, horizontal to right.
upperLeft	Vertical to top, horizontal to left.
upperCenter	Vertical to top, horizontal to center.
upperRight	Vertical to top, horizontal to right.
lowerLeft	Vertical to bottom, horizontal to left.
lowerCenter	Vertical to bottom, horizontal to center.
lowerRight	Vertical to bottom, horizontal to right.

upperLeft	upperCenter	upperRight
centerLeft	center	centerRight
lowerLeft	lowerCenter	lowerRight

x	X position of the tool (in screen coordinates). This property is read-only.
y	Y position of the tool (in screen coordinates). This property is read-only.
width	Width of the tool (in screen dimensions). This property is read-only.
height	Height of this tool (in screen dimensions). This property is read-only.
dagLx	Least x coordinate specified in the graph. This property is read-only.
dagLy	Least y coordinate specified in the graph. This property is read-only.

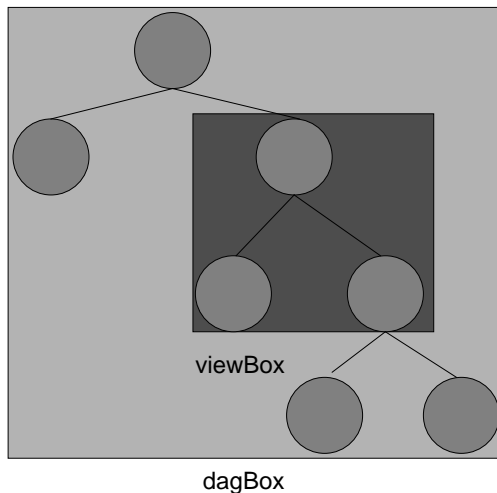
Cadence User Interface SKILL Functions Reference

Graph Browser

`dagUx` Greatest x coordinate specified in the graph. This property is read-only.

`dagUy` Greatest y coordinate specified in the graph. This property is read-only.

The view properties specify the box that the *dagTool* displays.



`viewLx` Lower left x coordinate that is displayed

`viewLy` Lower left y coordinate that is displayed.

`viewUx` Upper right x coordinate.

`viewUy` Upper right y coordinate.

`textOnly` Tells the *dagTool* to draw the nodes as text boxes, rather than using displayLists. The default is `nil`. The text is displayed in a raster font.

`showLabels` Turns on label display mode in the tool. The *dagTool* displays the *label* property of the *dagNode*.

`labelJustification` Specifies where the label is displayed relative to the *dagNode* when the *showLabels* property is `t`. It follows the same syntax and behavior as the *anchorTo* property.

`fontName` Specifies which raster font is used to display the labels when the *showLabels* property is `t`.

Actions and Action Lists

The ability to construct and display a directed graph is sufficient for many applications. In addition, however, the dag interface allows you to put actions on a class or node. Actions are SKILL procedures this node or nodes of this class can invoke using pop-up menus. When a node is displayed in the browser tool and has actions associated with it, you can press the middle mouse button over the node and a menu of actions is displayed.

Actions are assigned to a node or class using a list. This list is of the form:

```
( t_actionName t_actionProcedure t_actionArgs [t_actionAccelerator]  
  [g_enabled] )
```

<i>t_actionName</i>	The name of the action as it should appear in the popup menu.
<i>t_actionProcedure</i>	The name of the SKILL procedure called when the named action is selected.
<i>t_actionArgs</i>	String specifying all the arguments passed to the actionProcedure or the empty string ("") if there are none. This string and the action procedure name are combined with sprintf to construct the expression evaluated when the action is invoked.
<i>t_actionAccelerator</i>	Accelerator string specifying which key or mouse button will trigger this action. Valid actionAccelerator values are as follows: <ul style="list-style-type: none">■ any single alphanumeric (a-z, A-Z, 0-9) character (invoked by using the equivalent key from the keyboard)■ "mouseLeft" (invoked by a single click of the left mouse button)■ "mouseRight" (invoked by a single click of the right mouse button)■ "mouseLeft2" (invoked by a double click of the left mouse button)■ "mouseRight2" (invoked by a double click of the right mouse button)
<i>g_enabled</i>	[t or nil] Specifies whether the action is enabled (t) or disabled (nil). Disabled actions are displayed in gray in a popup menu and their action accelerators are not enabled.

Cadence User Interface SKILL Functions Reference

Graph Browser

For example the action specified by

```
("Hello" "println" "\"hello\"" " " t)
```

is invoked by the browser as

```
println("hello")
```

The action specified by

```
("Add" "addTwoNumbers" "1 3" "a" t)
```

is invoked by the browser as

```
addTwoNumbers( 1 3 )
```

Pressing the *a* key (while the cursor is positioned over a dagNode with this accelerator) also triggers this action.

One cautionary note about the action menus is that the popup menu is created the first time the user presses the middle mouse button over the node. No changes to the *actionList* will be reflected after the popup is created.

Note that the context-sensitive bind keys (action accelerators) supersede the ordinary *hi* interface bind key functionality.

dag Functions

dagAddActionToObject

```
dagAddActionToObject(  
    l_action  
    w_object  
)  
=> t_actionName / nil
```

Description

Adds the action to the specified dag node or class. However, if the action is already in the list, it is not replaced. For an existing action, use `dagDeleteActionFromObject` to delete the action and then call `dagAddActionToObject`.

Nodes inherit the actions of their class.

Arguments

<i>l_action</i>	List specifying the action. See “Actions and Action Lists” on page 777 for more information.
<i>w_object</i>	Node or class the action is applied to.

Value Returned

<i>t_actionName</i>	The name of the action (car of the action list).
<i>nil</i>	The function was not successful.

Example

To add an action to the myClass *dagClass*, enter

```
action = '("Hello" "println" "\"hello\"")  
dagAddActionToObject(action myClass)
```

dagCreateClass

```
dagCreateClass(  
    t_className  
)  
=>w_class / nil
```

Description

Creates a *dagClass* and returns it. If a class by this name already exists, it is overwritten.

Arguments

<i>t_className</i>	Name of this class. Make the name of each class unique among the names of classes used in any dag browser tool. This is probably best done by adding a unique prefix to the class name.
--------------------	---

Value Returned

<i>w_class</i>	The new class.
<i>nil</i>	There was an error.

dagCreateNode

```
dagCreateNode(  
    t_nodeName  
    w_class  
)  
=> w_node / nil
```

Description

Creates a dagNode and returns it.

Arguments

<i>t_nodeName</i>	Name of the node. The name of each node must be unique among all its siblings.
<i>w_class</i>	Class of this node as returned by <u>dagCreateClass</u> .

Value Returned

<i>w_node</i>	The new node.
<i>nil</i>	There was an error.

dagDeleteActionFromObject

```
dagDeleteActionFromObject(  
    t_actionName  
    w_object  
)  
=> t / nil
```

Description

Deletes the action named *t_actionName* from the specified class or node. If a node's action was inherited from its class, the action must be deleted from the class, not the node.

Arguments

<i>t_actionName</i>	Name of the action.
<i>w_object</i>	Class or node the action is applied to.

Value Returned

t	The action was deleted.
nil	The action was not deleted.

dagDestroyNode

```
dagDestroyNode(  
    w_node  
)  
=> t / nil
```

Description

Destroys a *dagNode*. All descendant nodes and arcs of this node are also destroyed, as well as the arcs linking this node to its parents.

Arguments

w_node Node to be destroyed.

If you do not want the descendants of this node to be destroyed, unlink the node's immediate children with [dagUnlinkParentFromChild](#) before calling this function.

Value Returned

t The node was destroyed.

nil There was an error.

dagDisplayTool

```
dagDisplayTool(  
    [w_dagTool]  
    [g_runPlacer]  
)  
=> t / nil
```

Description

Redisplays a *dagTool*. You can use this function if you want to re-display all of the objects in the graph rather than calling `dagRefreshObject` for each one.

In general, this routine need not be called. The dag will automatically refresh all tools in which it detects changes. If you wish to force the redisplay at a specific point in a procedure, you can call this routine.

See [dagRefreshObject](#) for the circumstances in which the dag does not automatically redisplay.

Arguments

<i>w_dagTool</i>	<i>dagTool</i> to be displayed. If no tool is specified, the current tool is displayed.
<i>g_runPlacer</i>	[t or nil] Tells the <i>dagTool</i> whether it should re-run the placement algorithm for the graph (if t) or just redisplay the nodes where they are. The default is t.

Value Returned

t	The dagTool was redisplayed.
nil	There was an error.

dagGetCurrentObject

```
dagGetCurrentObject(  
    )  
=> w_object / nil
```

Description

Often, when writing action procedures or SKILL objects, you need to know the node for which the action was selected. `dagGetCurrentObject` does this. It takes no arguments and returns the current node, or `nil`.

This function is intended for use in the action functions and in display list SKILL objects. The current object is set by the dag routines under the following conditions:

- When each dagnode is drawn, that node is made current.
- When the user invokes an action routine of a node, that node is made current.
- When the currently active node is deleted, in which case `dagGetCurrentObject` will return `nil`.

`dagGetCurrentObject` returns the object where the pointer is located at the time the function executes. If you start a long procedure and then move the pointer to another object before `dagGetCurrentObject` executes, `dagGetCurrentObject` returns the new object, not the original one.

If you call other procedures from your action callback, you should call `dagGetCurrentObject` in the first procedure only, and then pass a variable to the subprocedures. Do not call `dagGetCurrentObject` multiple times in a callback “tree.”

Arguments

None.

Value Returned

<i>w_object</i>	The current node.
<code>nil</code>	There is no current node.

Cadence User Interface SKILL Functions Reference

Graph Browser

Example

```
procedure( myDagCallback( )
  prog( (curObject)
    curObject = dagGetCurrentObject( )
    subProc( curObject )
  )
)
procedure( subProc( curObject )
  ; whatever
)
```

dagGetCurrentTool

```
dagGetCurrentTool(  
    )  
=> w_dagTool / nil
```

Description

Returns the current *dagTool*, and `nil` if there is not a current *dagTool*. Note that the current *dagTool* is different from the current window. The current tool might not be the current window.

This function is intended for use in the action functions and in display list SKILL objects. Immediately before any action callback procedure is called, the current tool value is updated.

Also, before a tool is redrawn, the current tool is updated. If the current tool is closed (deleted), `dagGetCurrentTool` will return `nil`.

Arguments

None.

Value Returned

<code>w_dagTool</code>	The current <i>dagTool</i> .
<code>nil</code>	There is no current <i>dagTool</i> .

dagLinkParentToChild

```
dagLinkParentToChild(  
    w_parent  
    w_child  
)  
=> w_arc / nil
```

Description

Creates a *dagArc* that links the parent node to the child node. It returns the arc it creates. If an error occurs, it returns `nil`.

The automatic placement algorithm places the children of a parent node in the order in which they were added with this function.

Arguments

<i>w_parent</i>	Parent node to link from; head of the arc to be created.
<i>w_child</i>	Child node to link to; tail of the arc to be created.

Value Returned

<i>w_arc</i>	The new arc.
<code>nil</code>	There was an error.

dagNumToTool

```
dagNumToTool(  
    x_toolNumber  
)  
=> w_dagTool / nil
```

Description

Returns the *dagtool* that matches the window number specified by *x_toolNumber*.

Arguments

<i>x_toolNumber</i>	The <i>dagTool</i> number for which you are trying to get the associated <i>dagtool</i> .
---------------------	---

Value Returned

<i>w_dagTool</i>	The dagTool.
<i>nil</i>	The dagTool was not found.

dagOpenTool

```
dagOpenTool(  
    x_backGroundColor  
    l_bBox  
    w_startingObject  
    t_name  
    t_help  
)  
=>w_dagTool / nil
```

Description

Creates a *dagTool* object and opens a window in which to display the graph starting from the specified starting node.

Arguments

<i>x_backGroundColor</i>	The background color of the tool. Specifying -1 for this argument is the same as passing <code>hivEditorBackgroundColor</code> , which means the window will use the editor background color.
<i>l_bBox</i>	List specifying the lower-left and upper-right coordinates of the tool (in screen coordinates). If this list is <code>nil</code> , you are prompted to position the window on the screen.
<i>w_startingObject</i>	The node that the tool uses to start processing from. This property can be changed after the tool is opened.
<i>t_name</i>	Name displayed in the dag browser tool's window banner.
<i>t_help</i>	A string giving the name of a symbol for the help message for this tool. It determines what is shown when you press the Help button in the dag browser tool's banner line.

Value Returned

<i>w_dagTool</i>	The <i>dagTool</i> that was opened.
<i>nil</i>	The <i>dagTool</i> could not be opened.

Cadence User Interface SKILL Functions Reference

Graph Browser

Example

```
tool = dagOpenTool( hiMatchColorByName( "white"  
list(100:100 400:400)  
myNode "myTool" "xxxHelpSymbol" )
```

Closing a dagTool

To close a dagTool, call `hiCloseWindow`. For example

```
hiCloseWindow( tool->window )
```

If you have specified a *closeProc* property on the tool, that procedure is called just before the tool or the window for it is destroyed. The user may close a dag tool by using a window manager *Quit* or *Close* button, which will also cause the *closeProc* to be called.

In *closeProc* you should be sure to destroy all the nodes and arcs you created for this graph that you no longer need. For example, if the nodes in this browser tool are used only in this browser and you have no nodes pushed on the starting object stack, you could call

```
dagDestroyNode( tool->startingObject )
```

dagPopTool

```
dagPopTool(  
    w_dagTool  
)  
=> t / nil
```

Description

Pops the tool stack (specified by the *w_dagTool* argument).

The *dagTool* can have a stack of starting nodes. You can push new starting nodes onto the stack and pop them off. This is similar to the Layout Editor's Push and Return commands. There is no limit to the number of nodes you can have on the stack. Each time that you push or pop the stack, the tool is redisplayed, so there is no need to call `dagDisplayTool`.

Arguments

<i>w_dagTool</i>	The tool for which the starting node stack is being manipulated.
------------------	--

Value Returned

t	The function was successful.
nil	There was an error.

dagPushTool

```
dagPushTool(  
    w_dagTool  
    w_newStartingNode  
)  
=> t / nil
```

Description

Pushes a new starting object (specified by the *w_newStartingNode* argument) onto the stack of the tool (specified by *w_dagTool*).

The *dagTool* can have a stack of starting nodes. You can push new starting nodes onto the stack and pop them off. This is similar to the Layout Editor's Push and Return commands. There is no limit to the number of nodes you can have on the stack.

Arguments

<i>w_dagTool</i>	The tool for which the starting object stack is being manipulated.
<i>w_newStartingNode</i>	The new starting node to put on the <i>dagTool</i> 's stack.

Value Returned

<i>t</i>	The function was successful.
<i>nil</i>	There was an error.

dagRefreshObject

```
dagRefreshObject(  
    w_dagNode  
)  
=> t / nil
```

Description

Redraws the node (specified by *w_dagNode*) in all the tools it is currently displayed in.

`dagRefreshObject` should be called to ensure that the display is up to date if

- The node's display list has been changed (either by changing the node's display list directly or by changing the display list inherited from the class)
- The display list for the node contains a SKILL object which needs to be reevaluated
- The width, height, or display list properties inherited from the class have changed

Arguments

<i>w_dagNode</i>	The node to be refreshed.
------------------	---------------------------

Value Returned

t	The function was successful.
nil	There was an error.

dagSetActionStatus

```
dagSetActionStatus(  
    t_actionName  
    w_dagObject  
    g_status  
)  
=> t / nil
```

Description

Sets the status of the specified action in the action list of the class or node. An action that has been turned off is grayed out in the pop-up menu and not available through its action accelerator, if any.

Arguments

<i>t_actionName</i>	Name of the action.
<i>w_dagObject</i>	Node or class the action is associated with.
<i>g_status</i>	[<i>t</i> or <i>nil</i>] <i>t</i> turns on the action; <i>nil</i> turns it off.

Value Returned

<i>t</i>	The function was successful.
<i>nil</i>	There was an error.

dagSetCurrentTool

```
dagSetCurrentTool(  
    w_dagTool  
)  
=> t / nil
```

Description

Sets the current *dagTool* to the one specified by the *w_dagTool* argument.

Arguments

<i>w_dagTool</i>	The tool to make current.
------------------	---------------------------

Value Returned

t	The function was successful.
nil	There was an error.

dagSetExpandedActionStatus

```
dagSetExpandedActionStatus(  
    t_actionName  
    w_dagObject  
    g_status  
)  
=> t / nil
```

Description

Sets the status of the specified action in the expanded-action list of a node. Setting the status to `nil` makes the action name appear gray in the pop-up menu and makes the action accelerator unavailable. Once a *dagNode* has been placed into a tool, the actions from the *dagClass* are concatenated to make a full list of actions for this node. This is known as the expanded-action list.

Arguments

<i>t_actionName</i>	Name of the action.
<i>w_dagObject</i>	Node the action is associated with.
<i>g_status</i>	[<code>t</code> or <code>nil</code>] <code>t</code> enables the action; <code>nil</code> disables it.

Value Returned

<code>t</code>	The function was successful.
<code>nil</code>	There was an error.

dagSetFont

```
dagSetFont(  
    t_XFontName  
)  
=> t / nil
```

Description

Modifies the font used in text mode in any *dag* tool.

Arguments

<i>t_XFontName</i>	Name of the X font to be used in text mode.
--------------------	---

Value Returned

<i>t</i>	Returns <i>t</i> if the specified font is found.
<i>nil</i>	Returns <i>nil</i> if the specified font cannot be found.

dagUnlinkParentFromChild

```
dagUnlinkParentFromChild(  
    w_arc  
)  
=> t / nil
```

Description

Unlinks a parent node from a child node. It takes, as its only argument, the *dagArc* that connects the parent to the child. The arc is destroyed; however, neither the parent node nor the child node is destroyed. This can be useful if you want to change the parent of a node.

Arguments

<i>w_arc</i>	The arc connecting the parent and child.
--------------	--

Value Returned

<i>t</i>	The nodes were unlinked.
<i>nil</i>	There was an error.

Cadence User Interface SKILL Functions Reference
Graph Browser

Interprocess Communication

This appendix lists the old `hi` interprocess communication functions. These `hi` functions have been replaced by `ipc` functions. For information about `ipc` functions, see Chapter 1, “Interprocess Communication SKILL Functions,” of the *Cadence Interprocess Communication SKILL Functions Reference*.

hiActivateBatch

The `hiActivateBatch()` function has been replaced by the [ipcActivateBatch](#) function.

hiActivateMessages

The `hiActivateMessages()` function has been replaced by the [ipcActivateMessages](#) function.

hiBatchProcess

The `hiBatchProcess()` function has been replaced by the [ipcBatchProcess](#) function.

hiBeginProcess

The `hiBeginProcess()` function has been replaced by the [ipcBeginProcess](#) function.

hiCloseChild

The `hiCloseChild()` function has been replaced by the [ipcCloseProcess](#) function.

hiContChild

The `hiContChild()` function has been replaced by the [ipcContProcess](#) function.

hiGetExitStatus

The `hiGetExitStatus()` function has been replaced by the [ipcGetExitStatus](#) function.

hiGetPid

The `hiGetPid()` function has been replaced by the [ipcGetPid](#) function.

hiGetPriority

The `hiGetPriority()` function has been replaced by the `ipcGetPriority` function.

hilsActiveChild

The `hiIsActiveChild()` function has been replaced by the `ipcIsActiveProcess` function.

hilsAliveChild

The `hiIsAliveChild()` function has been replaced by the `ipcIsAliveProcess` function.

hiKillAllProcs

The `hiKillAllProcs()` function has been replaced by the `ipcKillAllProcesses` function.

hiKillChild

The `hiKillChild()` function has been replaced by the `ipcKillProcess` function.

hiReadChild

The `hiReadChild()` function has been replaced by the `ipcReadProcess` function.

hiSetPriority

The `hiSetPriority()` function has been replaced by the `ipcSetPriority` function.

hiSetSkillDesc

The `hiSetSkillDesc()` function has been replaced by the `ipcSkillProcess` function.

hiSkillProcess

The `hiSkillProcess()` function has been replaced by the `ipcSkillProcess` function.

hiSleep

The `hiSleep()` function has been replaced by the `ipcSleep` function.

hiSoftInterrupt

The `hiSoftInterrupt()` function has been replaced by the `ipcSoftInterrupt` function.

hiStopChild

The `hiStopChild()` function has been replaced by the `ipcStopProcess` function.

hiWait

The `hiWait()` function has been replaced by the `ipcWait` function.

hiWaitForChild

The `hiWaitForChild()` function has been replaced by the `ipcWaitForProcess` function.

hiWriteChild

The `hiWriteChild()` function has been replaced by the `ipcWriteProcess` function.

Strokes

This appendix describes functions related to strokes. Functions are listed alphabetically.

Overview

A stroke is a unique shape that users draw in a window by pressing and holding a mouse button (typically the right button). Each stroke invokes a command. Files of predefined strokes are shipped with the Cadence software and must be installed.

Strokes are most commonly used by the Virtuoso layout editor and the Preview floorplanning tool. For more information, see Chapter 3, “Using Layout Editor Commands,” in the *Virtuoso Layout Editor User Guide* or Appendix D, “Keyboard and Mouse Bindings,” in the *Preview Fundamentals Reference*.

Cadence User Interface SKILL Functions Reference

Strokes

hiGetStrokeBBox

```
hiGetStrokeBBox( )  
=> l_boundingBox / nil
```

Description

Returns the bounding box of the stroke when within a callback initiated by a stroke.

Arguments

None.

Value Returned

<i>l_boundingBox</i>	Bounding box of the stroke in user units.
<i>nil</i>	There is no current window or no valid application associated with the current window.

Reference

[hiGetStrokeFirstPt](#), [hiGetStrokeLastPt](#), [hiStroke](#)

hiGetStrokeFirstPt

```
hiGetStrokeFirstPt( )  
=> l_point / nil
```

Description

Returns the point location of the first pixel of a stroke when within a callback initiated by a stroke.

Arguments

None.

Value Returned

<i>l_point</i>	First pixel of the stroke, expressed in user units.
nil	There is no current window or no valid application associated with the current window.

Reference

[hiGetStrokeBBox](#), [hiGetStrokeLastPt](#), [hiStroke](#)

hiGetStrokeLastPt

```
hiGetStrokeLastPt( )  
=> l_point / nil
```

Description

Returns the point location of the last pixel of a stroke when within a callback initiated by a stroke.

Arguments

None.

Value Returned

<i>l_point</i>	Last pixel of the stroke, expressed in user units.
nil	There is no current window or no valid application associated with the current window.

Reference

[hiGetStrokeBBox](#), [hiGetStrokeFirstPt](#), [hiStroke](#)

Cadence User Interface SKILL Functions Reference

Strokes

hiStroke

```
hiStroke(  
    [l_point]  
)  
=> t / nil
```

Description

Mouse draw-thru callback that initiates a stroke command.

This command is bound to the right mouse button draw-thru when the stroke files are loaded (see *Virtuoso Layout Editor User Guide*). This command only works if the strokes files are loaded.

Arguments

<i>l_point</i>	Starting point for the stroke.
----------------	--------------------------------

Value Returned

<i>t</i>	Returns <i>t</i> if the stroke is not canceled.
----------	---

<i>nil</i>	Returns <i>nil</i> if there is no current window, if there is no valid application for the window, or if the stroke is canceled.
------------	--

Reference

[hiGetStrokeBBox](#), [hiGetStrokeFirstPt](#), [hiGetStrokeLastPt](#)

Cadence User Interface SKILL Functions Reference
Strokes

Index

Numerics

1D forms [211](#)
 2D forms [212](#)
 2D menus
 creating [124](#)
 verifying [154](#)

A

absolute screen coordinates [474](#)
 acceptNumber enterfunction flag [559](#)
 acceptString enterfunction flag [559](#)
 accessing form data [209](#)
 actions and action lists [777](#)
 active window [423](#)
 adding
 menu items [123](#), [153](#)
 menus, fixed [477](#)
 objects to display lists [700](#)
 addPoint function [610](#)
 addPointProc callback procedure [557](#)
 alwaysMap enterfunction flag [559](#)
 application modal dialog box [171](#)
 application type, windows [471](#)
 Apply buttons [331](#)
 applyEnterFun function [617](#)
 arcs (in a graph) [762](#)
 attachments [214](#), [217](#), [221](#), [223](#), [229](#), [258](#),
 [370](#)

B

banners
 labels
 changing or adding [519](#)
 deleting [520](#)
 menus
 deleting [524](#)
 inserting [521](#)
 replacing [529](#)
 status, deleting [525](#)
 bindkey commands [532](#)
 bindkeys

 displaying [542](#)
 getting [540](#)
 inheritance [548](#)
 inherited from [549](#)
 prefix list [551](#)
 registering prefix [547](#)
 setting [533](#)
 setting multiple [538](#)
 boolean buttons, creating [239](#)
 borders [422](#)
 bounding box
 fields, creating [236](#)
 windows, getting [494](#)
 browsers, graph (dag) [762](#)
 button box fields, creating [243](#)
 buttons
 Apply [331](#)
 boolean, creating [239](#)
 Cancel [332](#)
 OK [335](#)
 standalone, creating [241](#), [264](#)

C

callback
 menu items, changing [160](#)
 procedures [557](#)
 return values [31](#)
 callbacks [30](#)
 Cancel buttons [332](#)
 cancelEnterFun function [615](#)
 cancelling, list boxes [191](#)
 changeEnterFun function [619](#)
 changeNextEnterFun function [622](#)
 changing
 menu items
 callback [160](#)
 text [161](#)
 window attributes [443](#)
 checkPoint [791](#)
 clearAllEnterFunctions function [623](#)
 closing procedure
 registering [447](#)
 unregistering [448](#)
 closing, windows [446](#)

Cadence User Interface SKILL Functions Reference

colorIndex function [200](#)
colors (of pens) [728](#)
Command Interpreter Window [33](#)
composite object [29](#)
coordinate systems [764](#)
creating
 2D menus [124](#)
 browser tool [762](#)
 buttons
 boolean [239](#)
 standalone [241](#), [264](#)
 dialog boxes [171](#)
 display lists [697](#)
 fields
 bounding box [236](#)
 button box [243](#)
 cyclic [248](#)
 float [251](#)
 frame [255](#)
 integer [267](#)
 labels [271](#)
 layer [273](#)
 list [275](#)
 list box [278](#)
 multiline text [283](#)
 point [287](#)
 point list [290](#)
 radio [293](#)
 scale [304](#)
 separator [307](#)
 string [312](#)
 toggle [319](#)
 fixed menus, horizontal [125](#)
 form fields [210](#)
 forms
 options [286](#)
 standard [228](#)
 graphs [762](#)
 list boxes [193](#)
 menu items [129](#)
 menus [120](#), [128](#)
 fixed, vertical [138](#)
 pulldown [131](#)
 simple [133](#)
 windows [430](#), [434](#)
current window [422](#)
cursor
 getting position [344](#)
 setting insertion position [397](#)
cursor, moving to the end [364](#)
cyclic fields

adding new choices [220](#)
creating [248](#)

D

dag browser [762](#)
dag coordinates [764](#)
dagAddActionToObject function [779](#)
dagArc properties [766](#)
dagClass properties [767](#)
dagCreateClass function [780](#)
dagCreateNode function [781](#)
dagDeleteActionFromObject function [782](#)
dagDestroyNode function [783](#)
dagDisplayTool function [784](#)
dagGetCurrentObject function [785](#)
dagGetCurrentTool function [787](#)
dagLinkParentToChild function [788](#)
dagNode properties [769](#)
dagNumToTool function [789](#)
dagOpenTool function [790](#)
dagPopTool function [792](#)
dagPushTool function [793](#)
dagRefreshObject function [794](#)
dagSetActionStatus function [795](#)
dagSetCurrentTool function [796](#)
dagSetExpandedActionStatus
 function [797](#)
dagSetFont function [798](#)
dagTool properties [772](#)
dagUnlinkParentFromChild function [799](#)
deiconifying, windows [453](#)
deletePoint function [613](#)
deleting
 banners
 labels [520](#)
 status [525](#)
 fields [323](#)
 fixed menus [479](#)
 forms [324](#)
 labels, banners [520](#)
 menu items [141](#)
 menus [140](#)
 fixed [479](#)
 menus from banners [524](#)
 status banners [525](#)
 windows [446](#)
delPointProc callback procedure [558](#)
dialog boxes
 creating [171](#)

- displaying [175](#)
- samples [183](#)
- dimensions, windows [475](#)
- directed acyclic graph (dag) [762](#)
- disabling, menu items [142](#)
- display lists
 - adding objects to [700](#)
 - described [696](#)
 - drawing [716](#)
 - objects in [696](#)
 - pens [728](#)
 - viewing [737](#)
- display state, windows [449](#)
- displaying
 - bindkeys [542](#)
 - dialog boxes [175](#)
 - forms, standard [325](#)
 - menus
 - Edge [143](#)
 - fixed [145](#)
 - general [147](#)
 - popup [148](#)
 - windows [430](#), [437](#), [458](#)
- dlAddArc function [701](#)
- dlAddBox function [702](#)
- dlAddCircle function [703](#)
- dlAddDonut function [704](#)
- dlAddEventObject function [705](#)
- dlAddPath function [707](#)
- dlAddPoint function [708](#)
- dlAddPolygon function [709](#)
- dlAddRasterText function [710](#)
- dlAddSegment function [712](#)
- dlAddSkillObject function [713](#)
- dlAddStrokeText function [714](#)
- dlAttachDlistToWidget function [738](#)
- dlAttachDlistToWindow function [739](#)
- dlClearDisplayList function [740](#)
- dlCloseWidget function [741](#)
- dlConfigureButton function [742](#)
- dlDetachDlistFromWidget function [743](#)
- dlDetachDlistFromWindow function [744](#)
- dlDisplay function [745](#)
- dlDlistToIcon function [746](#)
- dlDrawArc function [717](#)
- dlDrawBox function [718](#)
- dlDrawCircle function [719](#)
- dlDrawDonut function [720](#)
- dlDrawPath function [721](#)
- dlDrawPoint function [722](#)
- dlDrawPolygon function [723](#)

- dlDrawRasterText function [724](#)
- dlDrawSegment function [726](#)
- dlDrawStrokeText function [727](#)
- dlEnableItem function [747](#)
- dlFitDlistOnDraw function [748](#)
- dlMakeDisplayList function [698](#)
- dlMakeDlistButton function [749](#)
- dlMakePenTable function [729](#)
- dlMakeStipple function [730](#)
- dlMakeWidget function [750](#)
- dlMapWidget function [751](#)
- dlMoveButton function [752](#)
- dlResizeButton function [753](#)
- dlSaveDlist function [754](#)
- dlSetCurrentPen function [731](#)
- dlSetDlistPosition function [755](#)
- dlSetDlistScale function [756](#)
- dlSetPenColor function [732](#)
- dlSetPenFillStyle function [733](#)
- dlSetPenStipple function [735](#)
- dlSetPenTable function [736](#)
- dlSetWidgetName function [757](#)
- dlUnMapWidget function [758](#)
- doneProc callback procedure [558](#)
- dontDraw enterfunction flag [558](#)
- dplp function [39](#)
- drawEnterFun function [631](#)
- drawing with display lists [716](#)

E

- editable, fields [387](#)
- enabling, menu items [150](#)
- encapsulation window [634](#)
- enterArc function [564](#)
- enterBox function [567](#)
- enterCircle function [571](#)
- enterDonut function [574](#)
- enterEllipse function [578](#)
- enterfunctions
 - flags [558](#)
 - nesting [559](#)
 - overview [554](#)
 - prompts [555](#)
 - terminating [556](#)
- enterLine function [581](#)
- enterMultiRep function [604](#)
- enterNumber function [584](#)
- enterPath function [586](#)
- enterPoint function [590](#)

Cadence User Interface SKILL Functions Reference

enterPoints function [593](#)
enterPolygon function [596](#)
enterScreenBox function [599](#)
enterSegment function [601](#)
enterString function [608](#)
envCyclicStringToIndex function [93](#)
envGetAvailableTools function [94](#)
envGetDefVal function [95](#)
envGetLoadedTools function [96](#)
envGetModifiedTools function [97](#)
envGetVal function [98](#)
envGetVarType function [99](#)
envIsToolModified function [100](#)
envIsVal function [101](#)
envLoadFile function [102](#)
envLoadVals function [103](#)
envRegLoadDumpTrigger function [104](#)
envRegSetTrigger function [105](#)
envSetVal function [108](#)
envStoreEnv function [111](#)
error dialog box [175](#)

F

field attachments [214](#), [217](#), [221](#), [223](#), [229](#),
[258](#), [370](#)
fields
 adding to forms [223](#)
 bounding box, creating [236](#)
 button box, creating [243](#)
 creating
 sample [409](#)
 cyclic
 adding new choices [220](#)
 creating [248](#)
 default, setting [396](#)
 deleting [323](#)
 dimensions, getting [341](#)
 editable, setting [387](#)
 float, creating [251](#)
 frame, creating [255](#)
 integer, creating [267](#)
 labels, creating [271](#)
 layer
 creating [273](#)
 current, setting [398](#)
 iconic layer choices [361](#)
 layer object, getting [345](#)
 list box
 creating [278](#)
 sample [414](#)
 setting the toplist item [401](#)
 list, creating [275](#)
 moving [363](#)
 multiline text [283](#)
 sample [418](#)
 multiline text, size [349](#)
 point list, creating [290](#)
 point, creating [287](#)
 radio, creating [293](#)
 resizing [369](#)
 scale, creating [304](#)
 separator, creating [307](#)
 setting off [367](#)
 sizing [369](#)
 string
 sample [407](#)
 string, creating [312](#)
 text, moving the cursor to the end [364](#)
 toggle, creating [319](#)
 type-in
 current [340](#)
 current, setting as [385](#)
 highlighting [352](#)
fill patterns [728](#)
finishEnterFun function [616](#)
fixed menu
 deleting [479](#)
fixed menus
 adding [477](#)
 displaying [145](#)
 Edge, setting [158](#)
 horizontal, creating [125](#)
 identifying [481](#)
 moving [480](#)
 overview [120](#)
 removing [479](#)
 sample [166](#)
flagging callback errors [31](#)
float fields, creating [251](#)
focusing windows [462](#)
formProc callback procedure [558](#)
forms
 accessing data [209](#)
 adding fields [223](#)
 attributes [213](#)
 changing the callback [226](#)
 creating [228](#)
 creating fields [210](#)
 current, getting [343](#)
 default fields, setting [396](#)

Cadence User Interface SKILL Functions Reference

- deleting [324](#)
- deleting fields [323](#)
- displaying, standard [325](#)
- layout [211](#)
- location
 - setting [394](#)
 - storing [405](#)
- one-dimensional (1D) [211](#)
- options
 - creating [286](#)
 - overview [209](#)
- samples [407](#)
- standard [208](#)
- titles, changing [227](#)
- two-dimensional (2D) [212](#)
- unmapping [338](#)
- frame fields, creating [255](#)

G

- getting
 - window menu [152](#)
 - window name [464](#)
- global symbol [29](#)
- graph browser [762](#)
- graph, directed acyclic [762](#)

H

- help button [422](#)
- hiAbsolutePan function [498](#)
- hiAddCyclicChoice function [220](#)
- hiAddField function [221](#)
- hiAddFields function [223](#)
- hiAddFixedMenu function [477](#)
- hiAddMenuItem function [123](#)
- hiAddTextWordDelimiter function [685](#)
- hiAppendInputCmd function [643](#)
- hicErrorDialog [175](#)
- hiChangeBannerLabel function [519](#)
- hiChangeFormCallback function [226](#)
- hiChangeFormTitle function [227](#)
- hiCheckAbort function [64](#)
- hicInformationDialog [175](#)
- hiCloseWindow function [446](#)
- hicMessageDialog [175](#)
- hicQuestionDialog [175](#)
- hiCreate2DMenu function [124](#)
- hiCreateAppForm function [228](#)

- hiCreateBBoxField function [236](#)
- hiCreateBooleanButton function [239](#)
- hiCreateButton function [241](#)
- hiCreateButtonBoxField function [243](#)
- hiCreateColorArray function [201](#)
- hiCreateCyclicField function [248](#)
- hiCreateFloatField function [251](#)
- hiCreateForm function [263](#)
- hiCreateFormButton function [264](#)
- hiCreateFrameField Function [255](#)
- hiCreateHorizontalFixedMenu
 - function [125](#)
- hiCreateIntField function [267](#)
- hiCreateLabel function [271](#)
- hiCreateLayerCyclicField function [273](#)
- hiCreateListBoxField function [278](#)
- hiCreateListField function [275](#)
- hiCreateMenu function [128](#)
- hiCreateMenuItem function [129](#)
- hiCreateMLTextField function [283](#)
- hiCreateOptionsForm function [286](#)
- hiCreatePointField function [287](#)
- hiCreatePointListField function [290](#)
- hiCreatePullDownMenu function [131](#)
- hiCreateRadioField function [293](#)
- hiCreateScaleField function [304](#)
- hiCreateScrollRegion function [257](#)
- hiCreateSeparatorField function [307](#)
- hiCreateSimpleMenu function [133](#)
- hiCreateSliderMenuItem function [136](#)
- hiCreateStringField function [312](#)
- hiCreateToggleField function [319](#)
- hiCreateVerticalFixedMenu function [138](#)
- hiCreateWindow function [434](#)
- hicWarningDialog [175](#)
- hicWorkingDialog [175](#)
- hiDBoxCancel function [173](#)
- hiDBoxOK function [174](#)
- hiDeiconifyWindow function [453](#)
- hiDeleteBannerLabel function [520](#)
- hiDeleteBannerMenu function [523](#)
- hiDeleteBannerMenus function [524](#)
- hiDeleteField function [322](#)
- hiDeleteFields function [323](#)
- hiDeleteForm function [324](#)
- hiDeleteMenu function [140](#)
- hiDeleteMenuItem function [141](#)
- hiDeleteStatusBanner function [525](#)
- hiDeltaPan function [497](#)
- hiDisableMenuItem function [142](#)
- hiDisableTailViewfile function [673](#)

Cadence User Interface SKILL Functions Reference

hiDisableTextSelDefault function [694](#)
hiDisplayAppDBox function [175](#)
hiDisplayBlockingDBox function [178](#)
hiDisplayEdgeMenu function [143](#)
hiDisplayFixedMenu function [145](#)
hiDisplayForm function [325](#)
hiDisplayListBox function [187](#)
hiDisplayMenu function [147](#)
hiDisplayModalDBox function [179](#)
hiDisplayModelessDBox function [180](#)
hiDisplayNonBlockingDBox function [181](#)
hiDisplaySysModalDBox function [182](#)
hiDisplayWindow function [437](#)
hiDisplayWindowMenu function [148](#)
hiEdgeFixedMenuDone function [149](#)
hiEditPropList function [327](#)
hiEnableMenuItem function [150](#)
hiEnableTailViewfile function [674](#)
hiEncap function [635](#)
hiEndLog function [69](#)
hiFixedMenuDown function [151](#)
hiFlush function [42](#)
hiFocusToCIW function [43](#)
hiFocusToCursor function [462](#)
hiFocusToEncap function [644](#)
hiFormApply function [331](#)
hiFormCancel function [332](#)
hiFormDefaults function [334](#)
hiFormDone function [335](#)
hiFormList function [337](#)
hiFormUnmap function [338](#)
hiGenTextIndex function [653](#)
hiGetAbsWindowScreenBBox
function [474](#)
hiGetAppType function [471](#)
hiGetAttention function [44](#)
hiGetBannerMenus function [526](#)
hiGetBBoxResource function [79](#)
hiGetBeepVolume function [49](#)
hiGetBindKey function [540](#)
hiGetBindKeyInheritAlias function [550](#)
hiGetBindKeyInheritRoot function [549](#)
hiGetBindKeyPrefixList function [551](#)
hiGetCIWindow function [51](#)
hiGetCommandPoint function [50](#)
hiGetCurrentCmd function [624](#)
hiGetCurrentField function [340](#)
hiGetCurrentForm function [343](#)
hiGetCurrentIndex function [671](#)
hiGetCurrentWindow function [472](#)
hiGetDbuPoint function [427](#)
hiGetDisplayName function [45](#)
hiGetDrawThruDelta function [428](#)
hiGetEncapSkillCmd function [639](#)
hiGetFieldInfo function [341](#)
hiGetFieldOverlaps function [342](#)
hiGetFont function [84](#)
hiGetGeometryResource function [80](#)
hiGetIconName function [466](#)
hiGetLayerCyclicValue function [345](#)
hiGetListBoxFieldFit function [188](#)
hiGetLogFileName function [70](#)
hiGetMaxScreenCoords function [473](#)
hiGetMultiClickTime function [46](#)
hiGetNumMenus function [527](#)
hiGetNumVisibleItems function [190](#)
hiGetPoint function [52](#)
hiGetScreenPoint function [429](#)
hiGetScreenSize function [47](#)
hiGetStringResource function [82](#)
hiGetStrokeBBox function [806](#)
hiGetStrokeFirstPt function [807](#)
hiGetStrokeLastPt function [808](#)
hiGetTextCharAtLoc function [689](#)
hiGetTextClass function [675](#)
hiGetTextDispLoc function [693](#)
hiGetTextFieldFit function [349](#)
hiGetTextIndexLoc function [692](#)
hiGetTextLineColumn function [691](#)
hiGetTextSelByLoc function [655](#)
hiGetTextSelection function [654](#)
hiGetTextSourceLength function [690](#)
hiGetTextWidth function [85](#)
hiGetTextWordDelimiter function [684](#)
hiGetTopListItem function [351](#)
hiGetUndoLimit function [505](#)
hiGetUserAbort function [65](#)
hiGetViewBBox function [494](#)
hiGetWidgetType function [469](#)
hiGetWindowFixedMenu function [481](#)
hiGetWindowIconifyState function [454](#)
hiGetWindowList function [463](#)
hiGetWindowMenu function [152](#)
hiGetWindowName function [464](#)
hiGetWindowState function [449](#)
hiGetWMOffsets function [475](#)
high-level objects [29](#)
highlight color, setting [677](#)
highlighting, type-in fields [352](#)
hiGraphicMode function [48](#)
hiHighlightField function [352](#)
hilconifyWindow function [452](#)

Cadence User Interface SKILL Functions Reference

hiIgnoreProp function [354](#)
hiInFormApply function [355](#)
hiInheritBindKey function [548](#)
hiInsertBannerMenu function [521](#)
hiInsertMenuItem function [153](#)
hiInstantiateForm function [356](#)
hils2DMenu function [154](#)
hilsForm function [357](#)
hilsFormDisplayed function [358](#)
hilsIcon function [155](#)
hilsInFieldCancel function [360](#)
hilsInstantiated function [359](#)
hilsMenu function [156](#)
hilsMenuItemEnabled function [157](#)
hilsMenuItemFilled function [528](#)
hilsWidgetType function [470](#)
hiLayerMatchCyclicStr function [361](#)
hiLayerStringToLPP function [362](#)
hiListBoxCancel function [191](#)
hiListBoxDone function [192](#)
hiListView function [501](#)
hiLowerWindow function [455](#)
hiMapWindow function [458](#)
hiMarkNonNestable function [627](#)
hiMatchColor function [202](#)
hiMatchColorByName function [204](#)
hiMoveField function [363](#)
hiMoveFixedMenu function [480](#)
hiMoveInsBarToEnd function [364](#)
hiMoveWindow function [460](#)
hiNextWinView function [504](#)
hiOffsetField function [365](#)
hiOffsetFields function [367](#)
hiOpenWindow function [430](#)
hiPan function [495](#)
hiPickWindow function [457](#)
hiPrevWinView function [503](#)
hiQueryFont function [86](#)
hiQuit function [53](#)
hiRaiseWindow function [456](#)
hiReattachField function [370](#)
hiRedo function [508](#)
hiRedraw function [499](#)
hiRefreshTextWindow function [681](#)
hiRegCloseProc function [447](#)
hiRegisterBindKeyPrefix function [547](#)
hiRegTimer function [54](#)
hiRemoveFixedMenu function [479](#)
hiRemoveTextWordDelimiter function [686](#)
hiRepeat function [55](#)
hiReplaceAllBannerMenus function [529](#)
hiReplaceTextWordDelimiter function [687](#)
hiReplayFile function [72](#)
hiResetAbort function [66](#)
hiResizeField function [369](#)
hiResizeWindow function [461](#)
hiRestoreView function [502](#)
hiSaveAsViewfile function [651](#)
hiSaveView function [500](#)
hiSaveViewfile function [650](#)
hiScaleBox function [56](#)
hiScrollRegion function [262](#)
hiScrollWindowBottom function [668](#)
hiScrollWindowDown function [666](#)
hiScrollWindowLeft function [663](#)
hiScrollWindowRight function [664](#)
hiScrollWindowToCurrentIndex
function [669](#)
hiScrollWindowToIndex function [670](#)
hiScrollWindowTop function [667](#)
hiScrollWindowUp function [665](#)
hiSelectTextByLoc function [657](#)
hiSetAbort function [67](#)
hiSetBeepVolume function [58](#)
hiSetBindKey function [533](#)
hiSetBindKeys function [538](#)
hiSetCallbackStatus function [384](#)
hiSetCurrentField function [385](#)
hiSetCurrentIndex function [672](#)
hiSetCurrentWindow function [485](#)
hiSetEncapHistory function [641](#)
hiSetEncapPrompt function [640](#)
hiSetEncapSkillCmd function [637](#)
hiSetFieldEditable function [387](#)
hiSetFilter function [74](#)
hiSetFilterOptions function [75](#)
hiSetFixedMenuSize function [158](#)
hiSetFont function [87](#)
hiSetFormHighlights function [391](#)
hiSetFormName function [393](#)
hiSetFormPosition function [394](#)
hiSetFormToDefaults function [396](#)
hiSetIconName function [467](#)
hiSetLayerCyclicValue function [398](#)
hiSetListItemCenter function [403](#)
hiSetListItemVisible function [402](#)
hiSetMenuItemCallback function [160](#)
hiSetMenuItemText function [161](#)
hiSetMultiClickTime function [59](#)
hiSetTextClass function [676](#)
hiSetTextHighlightColor function [678](#)
hiSetTextSelectAll function [658](#)

Cadence User Interface SKILL Functions Reference

hiSetTextSelection function [656](#)
hiSetTopListItem function [401](#)
hiSetUndoLimit function [506](#)
hiSetUserPreferences function [60](#)
hiSetViewfile function [649](#)
hiSetWindowAtts function [443](#)
hiSetWindowIcon function [468](#)
hiSetWindowMenu function [162](#)
hiSetWindowName function [465](#)
hiSetWinStyle function [445](#)
hiShowBindKeys function [542](#)
hiShowBindKeysByAppType function [544](#)
hiShowBindKeysByWindow function [546](#)
hiShowFieldBorders function [404](#)
hiShowListBox function [193](#)
hiStartGenTextIndex function [652](#)
hiStartLog function [77](#)
hiStoreFormLocation function [405](#)
hiStringToIcon function [205](#)
hiStroke function [809](#)
hiSwitchWindowType function [450](#)
hiSynchronize function [61](#)
hiTextDisplayString function [680](#)
hiTextWidth function [89](#)
hiToggleEnterForm function [629](#)
hiUndo function [507](#)
hiUnmapWindow function [459](#)
hiUnregCloseProc function [448](#)
hiUnselectText function [659](#)
hiUnselectTextAll function [662](#)
hiUnselectTextByLoc function [660](#)
hiUnselectTextClass function [661](#)
hiUpdate function [628](#)
hiUpdateTextSelectionColors function [682](#)
hiVectorPan function [496](#)
hiZoomAbsoluteScale function [493](#)
hiZoomIn function [490](#)
hiZoomOut function [491](#)
hiZoomRelativeScale function [492](#)

I

icon name
 getting [466](#)
 setting [467](#)
 windows, setting [467](#)
icon position, windows [454](#)
iconifying, windows [452](#)
icons
 color array [201](#)

 matching color [202](#), [204](#)
 overview [199](#)
 verifying [155](#)
 windows, setting [468](#)
information dialog box [175](#)
initProc callback procedure [557](#)
inserting
 banner menus [521](#)
 menu items [153](#)
integer fields, creating [267](#)
intrinsic property inheritance [764](#)

L

labels
 banners
 changing or adding [519](#)
 deleting [520](#)
 fields, creating [271](#)
layer cyclic fields
 creating [273](#)
 current, setting [398](#)
 iconic layer choices [361](#)
 layer object, getting [345](#)
laying out forms [211](#)
line patterns [728](#)
list box fields
 creating [278](#)
 sample [414](#)
 setting the toplist item [401](#)
list boxes
 cancelling [191](#)
 creating and displaying [193](#)
 overview [185](#)
 samples [196](#)
list fields, creating [275](#)
listing
 viewing parameters [501](#)
 window IDs [463](#)
location
 forms, setting [394](#)
 forms, storing [405](#)
log output [35](#)
lowering a window [455](#)

M

mapping, windows [458](#)
menu bar [422](#)

Cadence User Interface SKILL Functions Reference

menu items

- adding [123](#)
- changing callback [160](#)
- changing text [161](#)
- creating [129](#)
- deleting [141](#)
- dimensions, getting [341](#)
- disabling [142](#)
- enabled, verifying [157](#)
- enabling [150](#)
- inserting [153](#)
- slider, creating [136](#)

menus

- 2D
 - creating [124](#)
 - verifying [154](#)
- associating with windows [162](#)
- banners
 - inserting [521](#)
 - replacing [529](#)
- counting [527](#)
- creating [120](#), [128](#)
- deleting [140](#)
- deleting from banners [524](#)
- displaying [147](#)
- Edge
 - displaying [143](#)
 - removing [149](#), [151](#)
 - setting [158](#)
- fixed [120](#)
 - adding [477](#)
 - deleting [479](#)
 - displaying [145](#)
 - horizontal, creating [125](#)
 - identifying [481](#)
 - moving [480](#)
 - overview [120](#)
 - removing [479](#)
 - sample [166](#)
 - vertical, creating [138](#)
- popup
 - displaying [148](#)
 - overview [120](#)
 - sample [165](#)
- pulldown
 - creating [131](#)
 - overview [121](#)
 - sample [164](#)
- samples [163](#)
- simple, creating [133](#)
- slider

- sample [168](#)

- verifying [156](#)

- message dialog box [175](#)

- modal dialog boxes [171](#)

- modeless dialog boxes [171](#)

moving

- cursor to the end [364](#)

- fields [363](#)

- menus, fixed [480](#)

- windows [460](#)

- multiline text fields

- sample [418](#)

N

naming

- icons, windows [467](#)

- windows [465](#)

- nodes (in a graph) [762](#)

- noInfix enterfunction flag [559](#)

O

- OK buttons [335](#)

- one-dimensional forms [211](#)

options forms

- creating [286](#)

- enterfunctions [555](#)

- overview [209](#)

P

- parent window [422](#)

- pens (in display lists) [728](#)

- point fields, creating [287](#)

- point list fields, creating [290](#)

popup menus

- displaying [148](#)

- overview [120](#)

- sample [165](#)

- prefix list, bindkeys [551](#)

- preloading points [556](#)

- preXY function [612](#)

- property list editor [218](#)

pulldown menus

- overview [121](#)

Q

question dialog box [175](#)

R

radio fields, creating [293](#)

raising, windows [456](#)

recursion (to be avoided) [31](#)

redoing [508](#)

redrawing, windows [499](#)

registering

 bindkey prefix [547](#)

removing

 Edge menus [151](#)

 menus

 Edge [149](#)

 fixed [479](#)

 windows [459](#)

replacing, banner menus [529](#)

resizing

 fields [369](#)

 windows [461](#)

restoring

 viewing parameters [502](#)

 viewing parameters, before zoom or
 pan [503](#)

retrieving

 application type [471](#)

 icon name for windows [466](#)

 window name [464](#)

S

samples

 dialog boxes [183](#)

 forms [407](#)

 list boxes [196](#)

 menus [163](#)

saving, viewing parameters [500](#)

scale fields, creating [304](#)

screen coordinates

 overview [764](#)

 windows, getting [473](#)

scroll region fields [262](#), [340](#), [352](#), [385](#), [391](#)

scroll region fields<startrange> [257](#)

scrolling [34](#)

selecting, windows [457](#)

separator fields, creating [307](#)

setting off fields [367](#)

simple menus, creating [133](#)

sizing

 fields [369](#)

 windows [461](#)

SKILL functions, syntax conventions [26](#)

slider menu items, creating [136](#)

slider menus

 sample [168](#)

standard forms [208](#)

start-up file [29](#)

string fields

 sample [407](#)

string fields, creating [312](#)

switching, widget type [450](#)

system modal dialog box [171](#)

T

text fields

 multiline [283](#)

titles of forms, changing [227](#)

toggle fields, creating [319](#)

transcript log [35](#)

two-dimensional forms [212](#)

type-in fields

 current [340](#)

 current, setting as [385](#)

 highlighting [352](#)

U

undoing

 hiPrevWinView [504](#)

 last command [507](#)

 last undo [508](#)

 maximum number

 getting [505](#)

 setting [506](#)

undrawEnterFun function [630](#)

unmapping

 forms [338](#)

 windows [459](#)

user entry functions [554](#)

V

- verifying
 - 2D menus [154](#)
 - an opened window [441](#)
 - icons [155](#)
 - menu item enabled [157](#)
 - menus [156](#)
- vertical fixed menus [138](#)
- view function [647](#)
- viewfile
 - management [646](#)
 - windows [645](#)
- viewing display lists [737](#)
- viewing functions [489](#)
- viewing parameters
 - listing [501](#)
 - restoring [502](#)
 - restoring, before zoom or pan [503](#)
 - saving [500](#)

W

- warning dialog box [175](#)
- widget type
 - switching [450](#)
 - windows, getting [469](#)
 - windows, verifying [470](#)
- window function [439](#)
- window types [765](#)
- windowp function [441](#)
- windows [473](#)
 - application type, getting [471](#)
 - attributes, changing [443](#)
 - banner functions [517](#)
 - border width [422](#)
 - bounding box, getting [494](#)
 - closing [446](#)
 - closing procedure
 - registering [447](#)
 - unregistering [448](#)
 - creating [434](#)
 - creating and displaying [430](#)
 - current
 - getting [472](#)
 - setting [485](#)
 - deiconifying [453](#)
 - deleting [446](#)
 - dimensions, getting [475](#)
 - display state [449](#)
 - displaying [437](#), [458](#)
 - focus, setting [462](#)
 - functions [424](#)
 - graphics, switching to text [450](#)
 - icon name
 - getting [466](#)
 - setting [467](#)
 - icon position [454](#)
 - icon, setting [468](#)
 - iconifying [452](#)
 - ID [439](#)
 - list of IDs [463](#)
 - lowering [455](#)
 - management [422](#)
 - mapping [458](#)
 - moving [460](#)
 - name
 - getting [464](#)
 - setting [465](#)
 - naming [465](#)
 - number [422](#), [439](#)
 - offsets, getting [475](#)
 - raising [456](#)
 - redrawing [499](#)
 - removing [459](#)
 - resizing [461](#)
 - screen coordinates
 - absolute [474](#)
 - getting [473](#)
 - selecting [457](#)
 - sizing [461](#)
 - text, switching to graphics [450](#)
 - unmapping [459](#)
 - verifying [441](#)
 - viewing functions [489](#)
 - viewing parameters
 - listing [501](#)
 - restoring [502](#)
 - restoring, before zoom or pan [503](#)
 - saving [500](#)
- widget type
 - getting [469](#)
 - switching [450](#)
 - verifying [470](#)
- zooming in [490](#)
- zooming in or out
 - relative [492](#)
 - scale [493](#)
- zooming out [491](#)
- word delimiters [683](#)

Cadence User Interface SKILL Functions Reference

working dialog box [175](#)
wtypep function [440](#)

Z

zooming in or out
 relative [492](#)
 scale [493](#)
zooming in, windows [490](#)
zooming out, windows [491](#)