
Vicon Shogun Scripting Guide

Contents

About scripting with Shogun	26
Creating scripts	27
Running scripts	28
Reviewing executed scripts	29
About Vicon Shogun documentation	30
About HSL scripting with Shogun Post	31
Use the Shogun Post Script Editor	32
Open and run HSL scripts	36
Write and edit HSL scripts	48
Use HSL scripts to work with data	55
Run scripts from the command line	98
HSL command reference	99
Commands in alphabetical order	100
abs	101
acos	103
addDropListItem	105

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

addLayer	108
addListBoxItem	110
addListViewItem	113
addNamespace	117
addParameter	119
addScript	121
addScriptPath	123
addStaticParameter	125
addTab	127
addToClip	129
alignAxis	132
amcExportOptions	135
animRange	137
applInfo	140
asfExportOptions	142
asin	144
assert	146
atan	148
attach	150
autoCreateLabelingClusters	152
autoCreateSolver	154
autohideWindow	156
autoLabel	158
autoLabelOptions	161
autoVST	165
autoVSTOptions	166

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

axiomLabel	167
axiomLabelOptions	169
bakeData	172
bakeLengths	175
breakTangents	177
bvhExportOptions	179
c3dExportOptions	181
calcIntersection	185
calibrateCharacter	187
calibrateCharacterOptions	189
calibrateLabelingCluster	192
calibrateLabelingClusterOptions	193
calibrateProp	194
calibratePropOptions	195
cameraView	196
camerasView	202
clearLog	206
clearScriptPaths	208
client	210
closeScript	212
collapse	214
compareModules	216
copyClip	219
copyData	222
copyKeys	224
copyPattern	226

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

copyString	228
cos	230
cpExportOptions	232
create	234
createBoneVirts	237
createCheckBox	239
createColorPicker	242
createDir	245
createDropList	247
createForm	250
createGroupBox	253
createKey	256
createLabelingCluster	258
createListBox	259
createListView	262
createNumBox	266
createPropVST	269
createPropVSTOptions	270
createPushButton	271
createRadioButton	274
createReconstructorScript	277
createSceneScript	279
createShelfButton	282
createShelfGroup	285
createShelfSeparator	287
createShelfTab	289

createSkelScript	291
createSolvingSetup	293
createSolvingSetupFromLabelingSetup	294
createSolvingSetupOptions	295
createStaticBox	297
createTab	300
createTangents	303
createTextBox	305
createTimeBox	308
createWindow	311
cropClip	313
cross	316
cutKeys	318
dataHealthView	320
dataIssuesMap	321
delete	322
deleteAllDropListItems	324
deleteAllListBoxItems	326
deleteAllListViewItems	328
deleteAllTabItems	330
deleteCharacters	332
deleteClipData	334
deleteDropListItem	336
deleteFile	338
deleteListBoxItem	340
deleteListViewItem	342

deleteRedundant	344
deleteScriptPath	346
deleteShelfGroup	348
deleteShelfTab	350
deleteTabItem	352
deleteTangents	354
delGlobalVar	356
destroyWindow	358
dirChooser	360
dockWindow	361
dot	363
enableControl	365
exists	367
exit	369
extrapolateFingers	371
fabs	372
fastForward	374
fbxExportOptions	376
fbxImportOptions	380
fileChooser	382
fileClose	385
fileOpen	387
fillGaps	389
filter	392
findBadData	395
findDeviantKeys	398

findDropListItem	401
findGap	404
findListBoxItem	407
findNonRigid	410
findPlaneCrossing	412
findSelectedKeys	414
findTabItem	416
findTail	418
findUnlabeled	420
fixOcclusion	422
fixOcclusionOptions	424
flatten	426
floatWindow	428
formatTime	430
frameToSmppte	432
getActiveClip	435
getActiveLayer	437
getActiveShelfTab	439
getActiveTake	441
getActiveView	443
getActiveViewType	445
getAngleTo	447
getAnimEnd	449
getAnimStart	451
getAttached	453
getAttachedTo	455

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

getAutoSaveFile	457
getBooleanArrayProperty	459
getBooleanProperty	461
getChannels	463
getCheckBoxCheck	465
getChildren	467
getClipboardText	469
getClips	471
getColor	473
getColorPickerColor	475
getConstraintError	477
getConstraintOffset	479
getConstraintPos	481
getConstraintsThisIsSource	483
getConstraintsThisIsTarget	485
getContributingCameras	487
getControlEnabled	489
getControlPos	491
getControlText	493
getControlVisibility	495
getCount	497
getCurrentChar	499
getCurrentLabel	501
getCurrentScript	503
getDirList	505
getDistance	507

getDropListItem	510
getDropListSellItem	513
getEclipseActiveTrial	515
getEclipseAssociatedCalibration	517
getEclipseAssociatedSubjects	519
getEclipseMarkedTrials	521
getEditTool	523
getExtrapolateFingersAfterSolve	525
getFileExtension	526
getFileList	528
getFileLocation	531
getFileName	533
getFilePath	535
getFilePos	537
getFiles	539
getFileType	541
getFloatProperty	543
getFocus	545
getGaps	547
getGlobalBooleanVar	550
getGlobalFloatVar	552
getGlobalIntVar	554
getGlobalStringVar	556
getGlobalVarExists	558
getGlobalVectorVar	560
getIntArrayProperty	562

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

getIntProperty	564
getKeys	566
getLabelerChars	568
getLabelingClusterOffsets	570
getLastFile	571
getLastScript	573
getLayers	574
getLength	576
getListBoxItem	578
getListBoxSellItems	581
getListViewItemCheck	583
getListViewItemText	586
getListViewSellItems	589
getLogFile	592
getMarkerConnection	594
getMarkerConnectionColor	596
getModule	598
getModuleRange	600
getModules	602
getModuleType	604
getNamespace	606
getNamespaces	608
getNumBoxNum	610
getNumDropListItems	612
getNumKeys	614
getNumListBoxItems	616

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

getNumListViewColumns	618
getNumListViewItems	620
getNumModules	623
getNumShelfTabs	625
getParameter	627
getParameterExpression	629
getParameterPriorValue	631
getParameters	633
getParameterType	635
getParameterUserValue	637
getParent	639
getPlayEnd	641
getPlayStart	643
getPointClosestTo	645
getPosition	647
getPriority	649
getProfileInt	651
getProfileString	653
getProperty	655
getRadioButtonCheck	657
getRate	660
getRotation	662
getSavePath	664
getScale	666
getSceneName	668
getScriptPaths	670

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

getSelectedKeys	672
getSelectedTimeRanges	674
getSelectionFilter	676
getSelectionSetNodes	678
getSelectionSets	680
getSelectionSetSets	682
getSession	684
getShelfTabs	685
getSolverBones	687
getSolversFirst	689
getStringArrayProperty	691
getStringProperty	693
getSystemTime	695
getTabItem	697
getTabSellItem	699
getTime	701
getTimeBoxTime	703
getTimecode	705
getTimeCodeOffset	706
getTimecodeStandard	708
getTopLevelForm	710
getTrajectories	713
getUseAxiomSolving	715
getVectorProperty	717
getViewFilter	719
getViewLayout	720

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

getViewTypes	721
getWindowRect	723
getWindowVisibility	725
goToBasePose	727
goToPreferredPose	728
graphView	730
grayscalefitOptions	733
hasKey	734
hasParameter	737
help	739
hide	741
imageLabeler	743
insertBone	745
isAxiomEnabled	746
isEndOfFile	747
isFileBinary	749
isFileReadable	751
isFileWriteable	753
isSelected	755
isSelectionSet	757
kinematicLabelOptions	759
kinLabel	761
label	763
labelOptions	765
layoutForm	770
listCommands	773

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

listObjectTypes	775
listParameters	777
loadFile	779
loadScript	782
loadWorkspaceString	783
makeRigid	785
makeUnique	788
manipulator	789
markingMenu	793
master	794
mcpExportOptions	796
mcpImportOptions	798
messagePrompt	800
moveClip	803
moveRotKeyToPreRotCmd	806
moveTransKeyToPreTransCmd	807
multiplyJointRange	808
multiplyJointStiffness	810
newFile	812
normalize	814
offlineCameraHealthCheck	816
offlineDataProvider	819
offsetClips	820
openEclipseDatabase	822
pack	824
parent	826

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

pasteKeys	828
pathExists	830
play	832
playOptions	834
playRange	836
playSound	839
print	841
processFile	843
progressBar	844
python	848
quickPost	849
quickPostLabelOptions	851
quickPostOcclusionFixOptions	853
quickPostSolveOptions	854
readBool	855
readFloat	858
readInt	861
readLine	864
readRot	866
readString	869
readToken	872
readVec	874
readWord	877
reassemble	879
reconstruct	882
reconstructChunk	884

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

reconstructOptions	887
redo	891
reduceKeys	893
refreshEclipse	895
remoteControl	897
removeAllParameters	899
removeBone	901
removeFromClip	903
removeLayer	905
removeMarker	907
removeMarkerConnection	909
removeNamespace	911
removeParameter	913
removeRayContributions	915
removeSoftwareFitCentroids	917
removeUnusedParameters	919
renameParameter	921
renameShelfGroup	923
renameShelfTab	925
replaceNamespace	927
replay	929
resetClip	931
restoreGaps	933
rewind	935
rigidBody	937
rtDropFix	940

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

runScript	941
saveFile	943
saveScript	945
saveWorkspaceFile	947
scaleCharacter	949
sceneView	950
scriptExists	952
seekToString	954
select	956
selectabilityOn	959
selectBranch	961
selectByMarkerRadius	963
selectByName	965
selectByParameter	968
selectByPart	970
selectBySide	973
selectByType	975
selectChildren	977
selectClipObjects	979
selectDropListItem	982
selectionSet	984
selectKeys	987
selectListBoxItem	989
selectListViewItem	991
selectParent	994
selectProperty	996

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

selectRange	998
selectRelatedKeys	1001
selectSet	1003
selectShelfTab	1005
selectTabItem	1007
selectTails	1009
selectTree	1011
sendRemote	1013
server	1015
setActiveClip	1017
setActiveLayer	1020
setActiveTake	1023
setAltToSelect	1025
setAutoSaveFile	1027
setBoneLength	1029
setButtonShelfFile	1031
setCheckBoxCheck	1033
setCheckBoxHandler	1035
setColorPickerColor	1038
setColorPickerHandler	1040
setConstraint	1043
setControlAnchor	1045
setControlPos	1048
setControlText	1051
setControlTip	1053
setDir	1056

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

setDropListHandler	1058
setEditTool	1061
setErrorHandler	1063
setExitCode	1065
setExtrapolateFingersAfterSolve	1067
setFilePos	1068
setFocus	1070
setFrameRate	1072
setGlobalVar	1074
setHotKey	1075
setHotKeyFile	1079
setInterpType	1081
setKey	1083
setLanguage	1085
setLength	1086
setListBoxHandler	1088
setListViewColumns	1091
setListViewHandler	1094
setListViewItemCheck	1097
setListViewItemText	1100
setLogEnabled	1103
setLogFile	1104
setMarkerConnection	1106
setMarkingMenuFile	1109
setNumBoxHandler	1111
setNumBoxNum	1114

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

setNumBoxRange	1116
setParameter	1118
setParameterPrior	1120
setPinned	1122
setPosition	1123
setPreferredPose	1125
setPrimary	1126
setPriority	1128
setProperty	1130
setPushbuttonHandler	1133
setRadioButtonCheck	1136
setRadioButtonHandler	1138
setRangesFollowActiveClip	1141
setRotation	1143
setSavePath	1145
setScale	1147
setSceneName	1149
setScriptPaths	1151
setSelectionFilter	1153
setSelectionSetFile	1155
setSession	1157
setSolversFirst	1158
setStaticBoxHandler	1160
setStaticParameterExpression	1163
setStaticParameterUserValue	1165
setTabHandler	1167

setTextBoxHandler	1170
setTime	1173
setTimeBoxHandler	1175
setTimeBoxTime	1178
setTimeCodeOffset	1180
setTimeFormat	1182
setTrackList	1183
setUseAxiomSolving	1185
setViewFilter	1187
setWindowHandler	1188
setWindowSize	1191
show	1193
showControl	1195
showWindow	1197
sin	1199
skelSetup	1201
slave	1203
smpteToFrame	1205
snapTo	1207
snapToConstraint	1209
snapToLine	1212
snapToLocal	1214
snapToRigid	1217
snapToSystem	1220
snapToSystemAlign	1223
solve	1226

solver	1228
sortFloats	1231
sortInts	1233
sortStrings	1235
sqlColumnCount	1237
sqlColumnNames	1239
sqlColumnTypes	1241
sqlConnect	1243
sqlDisconnect	1245
sqlExecute	1247
sqlFirst	1249
sqlGetFloat	1251
sqlGetInt	1253
sqlGetRow	1255
sqlGetString	1257
sqlIsFieldNull	1259
sqlLast	1261
sqlNext	1263
sqlPrev	1265
sqlQuery	1267
squareRoot	1269
step	1271
stepKey	1273
stop	1275
strCompare	1277
strFind	1280

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

strLeft	1282
strLength	1284
strMid	1286
strReplace	1288
strReverseFind	1291
strRight	1293
strTok	1295
strTokArray	1298
swap	1301
system	1303
tabWindow	1306
tan	1308
testPrint	1310
tileClips	1311
tPoseLabel	1313
tPoseLabelOptions	1315
transcodeVideo	1317
trcExportOptions	1319
unbreakTangents	1321
uncalibrateLabelingCluster	1323
undo	1324
undoConsolidated	1326
unlabel	1327
unpack	1329
unparent	1331
unsetProperty	1333

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

updateClip	1335
velocityLabel	1337
velocityLabelOptions	1339
viewLayout	1342
vskExportOptions	1345
wait	1347
windowExists	1349
writeBool	1351
writeFloat	1354
writeInt	1357
writeProfileInt	1360
writeProfileString	1363
writeRot	1366
writeString	1369
writeVec	1372
x2dImportOptions	1375
xcplImportOptions	1377
zeroKey	1379
zoomView	1381
Commands by category	1383
Python scripting with Vicon Shogun	1404
Install the Shogun Post Python module	1405
Connect a Python client to Vicon Shogun Post	1406
Use Vicon Shogun Post SDK interfaces	1407
Switch the command line between HSL and Python ...	1408
Run Python scripts in Shogun Post	1409

Python / HSL interaction	1410
Run Python from ShogunPostCL	1411
Launch Python from the Start menu	1412

© Copyright 2017 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice.

Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Ltd.

Vicon® is a registered trademark of Oxford Metrics plc. Vicon Blade™, Vicon Control™, Vicon Lock™, Vicon Lock+™, Vicon Nexus™, Vicon MX™, Vicon Pegasus™, Vicon Shogun™, Vicon Studio™, T-Series™, Vicon Tracker™, Vicon Vantage™, Vicon Vero™, and Vicon Vue™, are trademarks of Oxford Metrics plc.

Bonjour is a trademark of Apple Inc., registered in the US and other countries.

Other product and company names herein may be the trademarks of their respective owners.

Vicon Motion Systems is an Oxford Metrics plc company. Email: support@vicon.com Web: <http://www.vicon.com>

About scripting with Shogun



This guide explains the scripting that is available with Shogun, introduces you to the script interface components, and teaches you the basic structure and tools you need to use scripts and to tailor them to your requirements.

Before you create scripts, ensure that you feel comfortable using Shogun Live and Post to capture, edit, and export data.

Note the following points:

- Almost all operations you perform manually in Shogun can be repeated using scripts.
- Shogun HSL scripts are based on the HOM Script Language (HSL).
- For information about Python scripting functionality, see [Python scripting with Vicon Shogun \(page 1404\)](#).

Creating scripts

- Most HSL scripts are created using the Shogun Post Script Editor.
- The easiest way to create an HSL script is to open the **Script Editor** (see [Open the Script Editor \(page 32\)](#)), click Record Actions , then perform one or more actions in Shogun. When you are done you can click Record Actions again to stop recording. When the steps are recorded, you can execute them by clicking Run Script , or customize them to create a more full-featured script.
- The **Script Editor** features tools for editing scripts, debugging scripts, saving scripts and loading previously saved scripts. For more information, see [Use the Shogun Post Script Editor \(page 32\)](#).
- You can use any ASCII text editor to create scripts as long as that editor uses only standard ASCII characters. Note that although you can create ASCII files with Microsoft® Word and WordPad, they frequently insert nonstandard characters like em-dashes and "curly" quotes, which Shogun cannot read.
- Sometimes you want to create scripts that prompts the user for information. You can do this in the following ways:
 - Create a user window that collects information which is subsequently passed to a script (see [Create user windows to run HSL scripts \(page 43\)](#)).
 - Create a pipeline operation. A pipeline operation is just a script with extra statements that ask the user to provide information prior to execution. You can string together multiple pipeline operations to create very sophisticated routines.

Tip

Remember that Shogun Post comes with example HSL scripts that you can use to modify and manipulate motion capture data. You can find these sample scripts in the *C:\Program Files\Vicon\ShogunPost#\Scripts* folder. Sometimes the easiest way to create a script is simply to duplicate and modify one of the existing scripts.

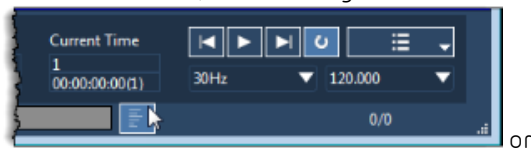
Running scripts

- You can execute HSL and Python commands in the command line on the status bar at the bottom of the Shogun Post window. This is handy when you just want to execute one or two commands and don't want to open a panel. For more information, see [Run scripts from the command line \(page 98\)](#).
- After you've created a script, you can create a hot key for it, add it to the **Marking** menu, or put it on the ribbon.

Reviewing executed scripts

- The **Log** displays a list of all the commands you have executed in the current session. To display the **Log**, either:

- On the status bar, click the Log button.



or

- On the **General** menu, click **Log**.
The **Log** displays a list of previously executed commands. This is useful when you want to see the results of previous commands while writing a script.
- You can also use the [print \(page 841\)](#) command to write the contents of specified strings to the **Log** (see [Print HSL command data to Log \(page 94\)](#)).

Now that you understand these fundamentals, you can begin creating, using, and allowing others to use your scripts.

About Vicon Shogun documentation

The following documentation is available with Shogun, both as help pages available online and as PDFs that you can download from docs.vicon.com:

Document	Description
<i>Installing and licensing Vicon Shogun</i>	Installation and licensing instructions.
<i>Getting started with Vicon Shogun</i>	Basic information on an end-to-end workflow for capturing data with Vicon Shogun Live and processing and exporting it with Vicon Shogun Post.
<i>Vicon Shogun Scripting Guide</i>	Scripting guidelines and commands.

For more documentation related to Shogun (for example, *PC Setup for Vicon Systems*) and other Vicon documents, visit the [Vicon website](https://www.vicon.com).

About HSL scripting with Shogun Post

For information about HSL scripting with Shogun Post, see the following topics:

- [Use the Shogun Post Script Editor \(page 32\)](#)
- [Open and run HSL scripts \(page 36\)](#)
- [Write and edit HSL scripts \(page 48\)](#)
- [Use HSL scripts to work with data \(page 55\)](#)

Use the Shogun Post Script Editor

The Vicon Shogun Post **Script Editor** is a handy pane that makes writing or editing Shogun HSL scripts easier. Use it to create, execute, and debug script commands to automate frequently used tasks or process complex data.

Use the buttons on the **Script Editor** toolbar for creating and managing HSL scripts, and to customize the display font and hot keys. Right-click and select commands for rapid access to basic **Script Editor** editing functions from the **Script Editor** shortcut menu.

For more information, see:

- [Open the Script Editor \(page 32\)](#)
- [Use the Script Editor toolbar \(page 32\)](#)
- [Use the Script Editor shortcut menu \(page 35\)](#)

Open the Script Editor

To display or hide the Script Editor:

Do one of the following:

- On the status bar, click the Script Editor button.



or

- Ribbon > Panels tab > Script Editor








Use the Script Editor toolbar

Use the toolbar at the top of the Script Editor to create and manage HSL scripts and customize the display font and hot keys.







Use the following commands and controls on the **Script Editor** toolbar:



Script management commands

Command		Description
New		Create a new script
Open		Open an existing script in the edit area below the toolbar
Close		Close the current script
Close All		Close all open files
Save		Save the current script
Save As		Save the current script as a new script file
Save All		Save all open files








Script creation and execution commands

Command		Description
Run		Run the current script.
Record Actions		Start recording actions to include in the current script. The clicks and keystrokes you make while recording get turned into Shogun Post commands.
Command Dialog		Display the Script Viewer , in which you can select from all the available Shogun Post commands. For a selected script, you can use the Copy Syntax , Copy Keyword , and or Edit Script buttons.
Run Selection		Execute just the selected portion of the current script. This is handy during debugging when you want to find out which statements work and which do not.

Customize Script Editor display and hot keys

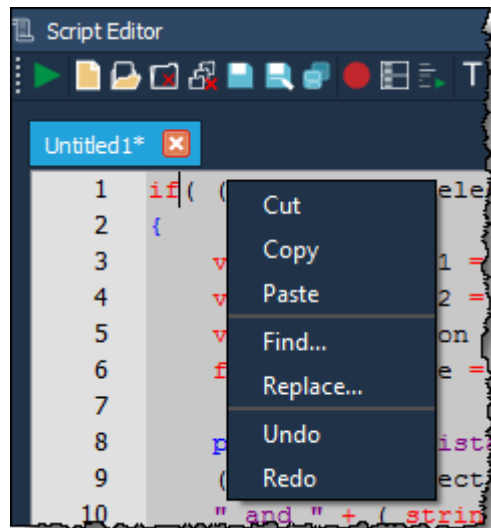
Command	Description
Change font 	Change the font family, style and size used to display scripts in the Script Editor .
Change hot keys 	Change the Script Editor hot keys assigned to listed commands or reset hot keys to their default settings.

Script debugging commands

Command	Description
Toggle Script Debugging 	Enter or exit debugging mode, which displays additional buttons that make debugging more advanced scripts easier: 
Toggle Breakpoint 	Set or clear breakpoints in the script
Clear Breakpoints 	Clear breakpoints in current script
Clear All Breakpoints 	Clear breakpoints in all scripts
Step Over 	Skip a specified breakpoint
Step Into 	Execute the command at a specified breakpoint

Use the Script Editor shortcut menu

Use the **Script Editor** shortcut (right-click) menu for rapid access to editing functions.



Use these commands on the **Script Editor** shortcut menu:

Command	Description
Cut	Cut the selected text from the Script Editor and transfer to the Windows clipboard.
Copy	Copy the selected text from the Script Editor to the Windows clipboard.
Paste	Paste text from the Windows clipboard to the location indicated by the cursor in the Script Editor .
Find	Locates selected text in the Script Editor . This function is case-sensitive.
Replace	Replaces selected Find text with the specified text.
Undo	Cancel the last Script Editor operation.
Redo	Reinstated the previous Undo operation.


Open and run HSL scripts

The following topics relate to opening and running Shogun Post HSL scripts:

- [Configure Shogun Post to locate HSL scripts \(page 36\)](#)
- [Load scripts from the HSL Script Editor \(page 42\)](#)
- [Undo and Redo HSL commands \(page 42\)](#)
- [Create user windows to run HSL scripts \(page 43\)](#)

You can also run both HSL and Python Shogun Post scripts from the command line (see [Run scripts from the command line \(page 98\)](#)).

Configure Shogun Post to locate HSL scripts


Shogun Post looks for HSL scripts in the *Scripts* folder (by default, *C:\Program Files\Vicon\ShogunPost#.#\Scripts*). Scripts in this folder can be selected from the **Pipelines** panel and the **Script Viewer**  (available from the **Script Editor** toolbar).

Shogun is supplied with a default scripts: a core set of generic scripts likely to be used by most Shogun users, as well as additional scripts that may be less commonly used but that illustrate what can be done in Shogun and how to script this behavior. You can modify these or write your own custom scripts to suit your requirements.

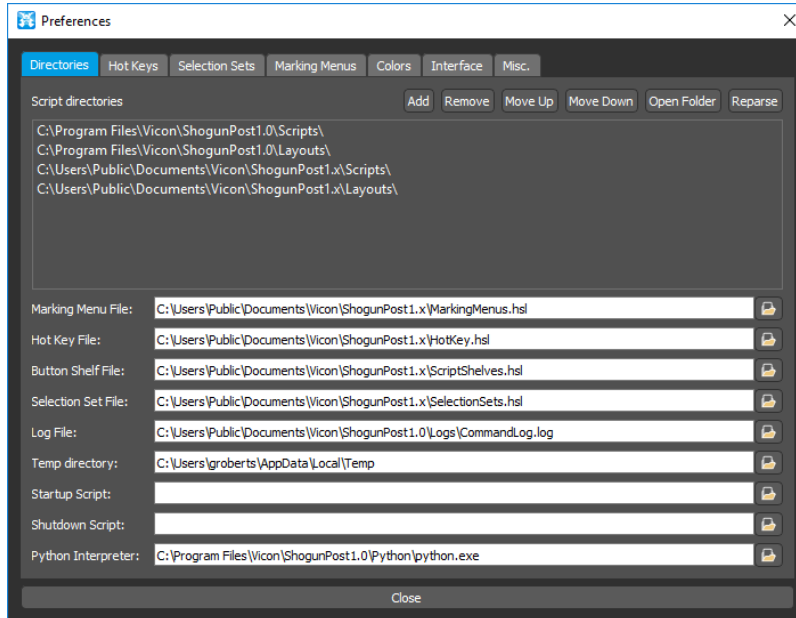
For information on specifying additional script folders, or changing the location of the button shelf script from the **Directories** tab in the **Preferences** dialog box, see the following topics:

- [Set Script folder locations \(page 36\)](#)
- [Set Button Shelf script file location \(page 39\)](#)
- [Default Scripts folder structure \(page 40\)](#)
- [Custom scripts directories \(page 42\)](#)

Set Script folder locations

The Shogun Post *Scripts* folder contains the default HSL scripts supplied with Shogun, organized by category into sub-folders. By default, the *Scripts* folder is installed under the main Shogun program folder (*C:\Program Files\Vicon\ShogunPost#.#\Scripts*). Scripts in this folder can be selected from the **Pipelines** panel and the **Script Viewer**  (available from the **Script Editor** toolbar). You can specify additional script folders in the **Preferences** dialog box, on the **Directories** tab:

General menu > Preferences option > Preferences dialog box > Directories tab



You can specify more than one folder here, but Shogun will execute the first version of a given script file that it finds. That means that if you have two script files named *greatScript.hsl*, Shogun will execute the first one it finds as it searches the specified script folders in the order in which they appear in this field. You can use this search order to manage when customized scripts are used over standard Shogun scripts.

Review the default scripts in the *Scripts* folder and all its sub-folders because they may save you time. Each script supplied with Shogun is documented, so you can quickly determine its function and modify its commands to meet your own needs. Most Shogun users find these scripts can streamline many standard operations as written, and they also can be customized to meet other needs as required.

Tip

Make sure you save a custom version of a Shogun script under a new file name. This ensures that if your script doesn't work, you still have the original.

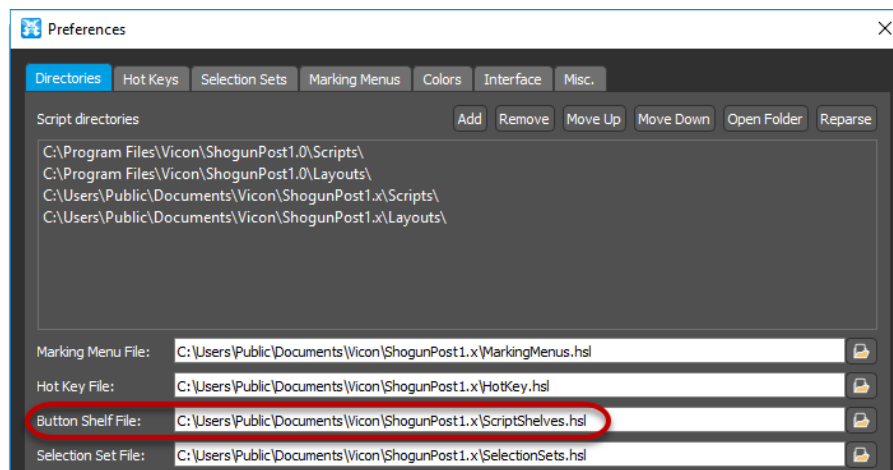
To specify additional script file locations:

1. From the **General** menu, open the **Shogun Post Preferences** dialog box, click the **Directories** tab, and view the **Script directories** section, which contains a row of buttons and a file list.
2. For each folder you want to add, in the **Script directories** mini toolbar, click the **Add** button and then in the **Folder** box, either type in the full path or click the **Browse** button and navigate to the desired location.
3. If you have specified multiple directories, use the **Move up** and **Move down** buttons in the mini toolbar to arrange the directories in the desired order, remembering that scripts are executed in the order listed.
4. To view the folder contents, select a folder in the list and click the **Open Folder** button on the mini toolbar.
5. To update the contents of the selected folder, click the **Reparse** button.

Set Button Shelf script file location

Shogun uses the *ScriptShelves.hsl* HSL script to build the Button Shelf interface. The button shelf tabs are displayed as custom tabs on the ribbon. By default, this script is located in the main ShogunPost program folder (*C:\Program Files\Vicon\ShogunPost#. #*). You can specify a different filename and/or location for this file on the **Directories** tab in the **Preferences** dialog box:

General menu > Preferences dialog box > Directories tab



You may find it useful to create a copy of the file and/or your own folder for storing a backup file. You can specify the location of this file and/or folder as described below.

To specify a different location for the Button Shelf script file:

1. From the **General** menu, open the **Preferences** dialog box.
2. On the **Directories** tab, in the **Button Shelf File** section, click the **Open** button at the end of the line.
3. In the **Open** dialog box, navigate to and select the *ScriptShelves.hsl* file that Shogun is to use, or supply a different name for the file and click **Open**.

Default Scripts folder structure

The Scripts folder (by default, *C:\Program Files\Vicon\ShogunPost#.#\Scripts*), contains a standard set of HSL scripts supplied with Shogun in the following structure:

Folder	Contents
\AutoPropVST	Deprecated script <i>AutoPropVST.hsl</i> , supplied for backward compatibility. See instead createPropVST (page 269) and createPropVSTOptions (page 270) .
\AutoSkeleton	<i>AutoSkeleton_MakeSkeleton.hsl</i> , a wrapper to enable createSolvingSetup (page 293) to be run as a pipeline operation.
\AutoVST	Script <i>AutoVST_MakeVST.hsl</i> .
\DefaultOps	Scripts that are set up specifically for use in a pipeline, though many other scripts can be used in pipelines.
\FileIO	Scripts relating to import to and export from Shogun Post.
\Interface	Scripts that control the 3D view, layout of views, hiding and showing of objects, etc.
\Labeling	Scripts that are for labeling markers.
\Modules	Scripts that operate on a single module, or collection of modules, often performing a low-level task:
	<div> \Attributes Scripts that modify generic attributes of a module. </div>
	<div> \Channels Scripts that operate on the channels of a module. This folder contains most of the functionality for modifying keys, marker data editing, and clips that are containers of keys. </div>
	<div> \Deletion Scripts that delete things. </div>
	<div> \Hierarchy Scripts that modify the parent-child relationship of modules. </div>
	<div> \Selection Scripts that select modules. </div>
\MXhardware	Scripts that operate on optical cameras and hardware units.

Folder	Contents
\NLE	Scripts that perform NLE operations like shifting, blending, looping, and adjusting clip start and end.
\Solving	Scripts that affect solving skeletons.
\SubjectCalibration	Scripts that calibrate subjects.
\Time	Scripts that change the current time or select time ranges.
\Tutorials	Scripts that are used in Shogun tutorials.
\Utilities	Scripts that are generic in use and can be used to create other scripts.
\Visualization	Scripts that are specific to visualizing something not easily visualized elsewhere in Shogun.



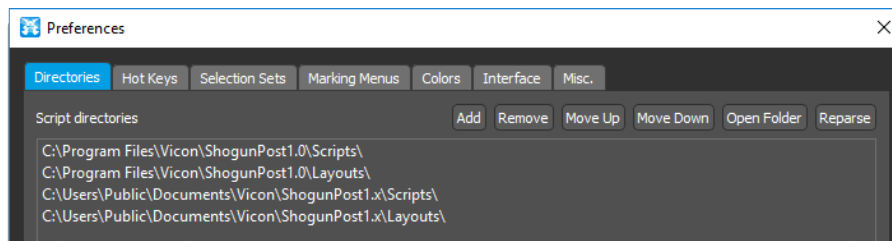
Important

The scripts that are supplied with Shogun Post are subject to change with each release.

Custom scripts directories

If you create customized scripts, we recommend that you to put these in a separate folder. You can call this folder anything and locate it anywhere. If you create customized scripts only for a single user, we recommend that you put these in a separate folder. It's a good idea to include the user name in the folder name.

Shogun Post searches for scripts in the same order that script folder locations are specified on the **Directories** tab in the **Preferences** dialog box. By putting customized scripts in separate folders from the standard Shogun Post *Scripts* folder and specifying their position, you can manage the priority in which scripts are applied. For example, if you have multiple scripts of the same name, Shogun executes the first version of a given script file that it finds. So, if you want to ensure that scripts customized for a specific user get used before a facility-wide version of that script, or the standard Shogun Post version, you would specify the following order: Personal, Facility, then *ShogunPost#. #\Scripts*.




Load scripts from the HSL Script Editor

Shogun Post ships with dozens of HSL scripts, which you can use and modify as required. These standard, ready-to-use scripts are contained in the *Scripts* folder, organized by category.

You can use scripts on different data files if objects called in the script exist in the current scene. When you execute a Shogun script, it does not become part of the scene; you must execute the script each time you want to repeat the action.

To load a Shogun Post script:

1. On the **Script Editor** toolbar, click the **Open** button .
2. In the **Open** dialog box, navigate to the directory containing the desired *.hsl* script file, and then click **Open**.

Undo and Redo HSL commands

You can use the undo and/or redo commands to return your scene file to its state immediately prior to the most recent command that affected data or keys.

By default, Shogun Post maintains a history stack of 20 operations. You can undo/redo a single step or by a specified number of operations up to 20. (To change this number, in the **Preferences** dialog box, click the **Misc** tab and alter the settings in the **Undo/Redo** section.)

Note that Shogun Post records only changes to data in its history stack; it does not record changes to the user interface or selected objects. Since such changes do not affect keys or data, they cannot generally be undone/redone because they have not changed any information.

For example, these commands can be undone because data changed:

```
| delete;  
| loadfile jump.hdf;
```

However, these commands cannot be undone because no data changed:

```
| select LHDF;  
| selectProperty Translation;
```

For further details, see [undo \(page 1324\)](#), [redo \(page 891\)](#), and [undoConsolidated \(page 1326\)](#).

Create user windows to run HSL scripts

You can customize the Shogun Post user interface by creating user windows that contain specified tools and commands.

Creating user windows to execute scripts can be useful in the following circumstances:

- | As you work on motion capture projects over time, you will develop a set of tools and routines you use frequently. Turning these tools into user windows will make them easier and faster to use.
- | If you must train, support, or work with other motion capture pros, creating custom user windows can accelerate production dramatically. Putting the tools people use most frequently at their fingertips, and insuring everyone uses the same tools, saves you time and money.

Any user windows you create in Shogun Post are just like the Shogun Post panels (docking windows) with the following exceptions:

- | Their docking information is not saved when Shogun Post shuts down. They must be recreated from a script every time Shogun Post restarts or any time they get destroyed.
- | They appear separately from the standard Shogun Post panels. They are appended as a new custom windows group to the end of the **Panels** tab on the ribbon.

User windows can hold any number of controls, the standard UI components you are familiar with, such as:

- | text boxes
- | check boxes
- | list boxes
- | number fields

All user controls are children of the user window they appear on. You can't parent a control to another control.

The following topics describe how to create user windows to run scripts:

- | [Create user windows and controls \(page 44\)](#)
- | [Set handlers \(page 45\)](#)
- | [Lay out windows and controls \(page 45\)](#)

If you are comfortable creating scripts, you should find creating your own user windows a straightforward process.

Create user windows and controls

All user windows and controls are identified by an integer value, called a handle. Shogun Post returns this handle whenever a user window or control gets created. This lets Shogun Post keep track of which window is accepting input from a user, and which control is being changed.

Save the handles associated with a window or control in int variables, and in your .ini file if you plan to call the window or control from any script other than one that created it.

The handle is an argument to almost every command in the user window category. For instance, to create a user window with a text box in it, you would write the following script:

```
// Variables to hold the handles
int $dlg;
int $textBox;
// Create the user window. Call it "Test User Window"
$dlg = `createWindow "Test User Window"`;
// Create the Text Box User Control, adding it to the
// User Window we just created
$textBox = `createTextBox $dlg`;
```

That's all that needs to be done to create a user window and add a control to it. Of course, to get any real functionality from the controls, you'll need to add a lot more scripting code.

Notice that the first argument or parameter required by the createTextBox command was the \$dlg variable which stores the handle for the window. The \$dlg variable tells the createTextBox command which user window to add the text box to.

The following commands provide functionality that is common to all user controls:

- | [showControl \(page 1195\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [setControlPos \(page 1048\)](#)
- | [getControlText \(page 493\)](#)
- | [setControlText \(page 1051\)](#)
- | [setControlAnchor \(page 1045\)](#)

For instance, showControl accepts as its first argument a handle for any user control, and it will show/hide/toggle any user type of user control.

Some controls (list boxes, drop lists, number boxes, etc.), have no functional use for setControlText. But for those controls, there are other commands that take its place. For instance, a number box doesn't really have text that is of any concern to the user. It is just the number value held in the control that the user wants. So instead of using getControlText /setControlText with those, you can just use [getNumBoxNum \(page 610\)](#)/[setNumBoxNum \(page 1114\)](#).

Set handlers

User windows and controls have scripts that get called when some event happens; these scripts are called "handlers".

For example, user windows have handlers for when the user changes the size of the window, when the user presses the Enter key when typing in a control in the window, or when the window receives an update notification from Shogun Post.

User controls all have handlers for events specific to their functionality. For instance, a drop list control has a selection changed handler that can get called any time the user changes what is selected in the list.

You can set the handler for this event by calling:

```
setListBoxHandler $listBoxID -selChange "HandlerScriptName";
```

where:

<code>\$listBoxID</code>	is the ID of a list box you created
<code>-selChange</code>	is the event you want to set a handler to
<code>"HandlerScriptName"</code>	is the name of your script (which must exist in your <i>Scripts</i> directory) that is called any time the list box selection changes.

These handlers are where you handle a lot of your interaction with the controls. For instance, in a user window, to accept customizable values for a bodyCleaner script, you might have a push button called "Run". You can set a handler for when that button gets clicked.

When it does, your handler script can suck values from all the other controls in the user window and use them in the bodyCleaner script.

Thus, you create a more user-friendly front end to the complex data manipulation scripts that you write.

Lay out windows and controls

Shogun Post uses a framework to allow controls to be placed and sized for whenever a user window is rescaled. This framework lets you keep sets of related controls together, and stretch controls to fit between the walls of other user controls or windows.

The "form" user control can be created just like any other user control. A form, however, is invisible. It has no visual representation other than the invisible rectangle which it occupies. A form is really a container for other controls that wish to be sized/placed relative to it. Controls can get added as sub items, or children, of a form.

✓ Tip

Each user window has a form called its top-level form, to which all other controls can be added. The top-level form is sized to the inner rectangle of the user window whenever the window changes. Thus, if you want controls to be aligned to the walls of the user window, create them as components of the top-level form of their user window, as shown in the following example.

```
// Variables to hold the handles
int $dlg;
int $textBox;
int $parentForm;
// This is the 4-element integer array, which represents the
// four corners of a rectangle which we'll use to size and
// place the controls.
// $rect[ 0 ] <--- LEFT
// $rect[ 1 ] <--- TOP
// $rect[ 2 ] <--- RIGHT
// $rect[ 3 ] <--- BOTTOM
int $rect[ 4 ];
// Create the user window. Call it "Test User Window" (can't be the
// same as any other docking windows)
$dlg = `createWindow "Second Test Window"`;

// Get a handle to the main user windows main layout form. All controls
// will be descendants of this and will be placed/sized relative to it.
$parentForm = `getTopLevelForm $dlg`;

// Create a static text box. Make it 80 pixels wide and 20 pixels high.
// Make it a sub-component of the user windows top level Form.
$rect[ 0 ] = 0; $rect[ 2 ] = 80;
$rect[ 1 ] = 0; $rect[ 3 ] = 20;
int $static = `createStaticBox $dlg -form $parentForm -pos $rect -text "Enter
Text Here:"`;

// Align 10 pixels from top side of the top level form, which in this
// case, is also the wall of the user window
setControlAnchor $static "top" "top" 10;
// Align 10 pixels from left side of the top level form, which in this
// case, is also the wall of the user window
setControlAnchor $static "left" "left" 10;

// Create a text box and also add it as a sub item of the Top Level
// Form. Have its top aligned with the top of the static text, and make
// it stretch from the right of the static text to the edge of the form.
$textBox = `createTextBox $dlg -form $parentForm`;
setControlAnchor $textBox "top" "top" 0 -target $static;
```

```
setControlAnchor $textBox "left" "right" 5 -target $static;  
setControlAnchor $textBox "right" "right" 10;  
  
// Finally, lay out all the controls with the anchors we added earlier  
layoutForm $parentForm;
```

In the example above, you create the user window and then have a handle to its top-level form. We stored the handle in a variable called `$parentForm`. Every time we created a control and wanted it to be sub item of the top-level form, we just added the `-form` option to the `createXXXX` command which takes a form user control's ID as an argument. Then we called `setControlAnchor` which sets the anchors for a control to the side of a form, or optionally, to the side of another target control.

The `setControlAnchor` command takes these four arguments:

- The control you wish to act on
- A string of which side of the control you want to be anchored
- The side of the parent form you wish to be anchored to This value should always be the same as the first if you are not targeting another control.
- An offset value which is in pixels, unless you specify the `-percent` flag, in which case it gets interpreted as a percent value (which must be positive if it's a percentage).

The `-target` flag allows you to specify another control's side to anchor to. So you can anchor the right side of the control to the left side of another control. Note that you can't anchor the top or bottom of a control to the left or right of another: that doesn't make any sense.

Also, forms can contain other forms, so you can, if you want, create a whole hierarchy of forms and controls. A couple of important notes about forms:

- When you call `showControl` or `enableControl` on a form, all sub components of the form get shown/hidden or enabled/disabled along with the form. So, for example, you have a group of controls that you want to hide all at once and are all placed relative to each other, just make them components of a form that itself is a sub component of the top-level form.
- Group boxes are not just user controls; they are also forms. Group boxes are the controls that look like rectangles with a text label on the top left. They have no real functionality other than visually demarcating groups of other controls. So, they are really just forms, and we made them such. But remember that forms are really just controls with extended functionality.

Write and edit HSL scripts

To write or edit HSL scripts, you'll need to learn how to do the following:

- [Understand HSL command syntax \(page 48\)](#)
- [Format HSL scripts \(page 51\)](#)

You can then edit scripts and create your own scripts (see [Use HSL scripts to work with data \(page 55\)](#)).

Understand HSL command syntax

You write Shogun Post commands using the following syntax:

```
command_name [ "command_argument" [ "..."] [-flag [flag_argument]] ;
```

where:

command_name	is the name of the Shogun Post command
["command_argument"]	is an optional variable name, parameter, value, or expression, that you must supply a value for.
...	Depending on the command, you might specify more than one argument. The square brackets [] indicate that this is an optional element; do not type these in your command. The quotation marks (" ") indicate that this is a variable; do not type these in your command.
[-flag]	is one or more optional flags that identify a status, condition, or event affecting the command. The dash (-) identifies the text as a flag; type this in front of the flag name with no intervening spaces.
[flag_argument]	Some flags may take their own arguments. Type a space between the flag and its argument. The square brackets [] indicate that this is an optional element; do not type these in your command.
;	is the special character required to indicate the end of the command. End each line of a script with a semicolon (;).


For more information, see:

- [Command capitalization \(page 49\)](#)
- [Basic command examples \(page 50\)](#)

For full syntax details for commands, see [HSL command reference \(page 99\)](#).



Tip

To see a quick reference for the syntax of any Shogun Post HSL command, on the Script Editor toolbar, click the Command Dialog button .

Command capitalization

All commands and flags start with lower case letters. If the command or flag name is a compound word, the first letter of the second word is always capitalized:

```
create;  
viewLayout;  
rigidBody;  
createShelfButton;  
create -addSelect;  
select -removePrimary;
```

Arguments are case sensitive. When you are defining object name arguments, you may want to adopt a convention you can remember easily, like all uppercase letters, all lower case letters, or the same kind of capitalization used in Shogun Post. Providing an argument or object name in the wrong case will cause a script to fail.

```
select -a Left_Thigh;  
select -all;
```

Basic command examples

The following examples show how the Shogun Post HSL command syntax is used in simple commands and scripts.

The syntax for the create command is:

```
create moduleType moduleName1 ["moduleName2"] ... [-parent] [-addSelect]
[-parent string] [-positionOffset vector] [-rotationOffset vector] [-
allClips]
```

where:

create	is the name of the command
moduleType	is an argument to specify the type of object to be created, a marker for example
"moduleName1"	is the name you want to call the new object
["moduleName2"] ...	means you can optionally provide the names for second and additional objects if desired
[-parent]	is an optional flag which specifies the parent of the new object so that you do not have to call the parent command separately after you run the create command. The -parent option takes one argument, the name /module path of the parent.
[-additional flags]	See create (page 234) for details of these.

To create a marker named leftPinky, you would type the following command:

```
create Marker leftPinky;
```

You can use multiple commands in a single script. Terminate each line of the script with a semicolon (;).

For example, in a single script you could use the create command to create a marker named leftPinky, the parent command to make it a node under the existing bone handBone, then select and setProperty commands to move the new marker down 45 units on the Translation Y axis:

```
create Marker leftPinky;
select -a handBone;
parent;
select leftPinky;
setProperty Translation -c 0 45 0;
```

Format HSL scripts

You can maximize your efficiency and reduce the time it takes you to complete projects by creating scripts. Most people use Shogun Post to repeatedly perform the same types of tasks. Taking three hours to create a great script for performing a task that would only take you half an hour is a good investment— if you use the script just six times, it has paid for itself. If you use it a hundred times you have saved yourself almost a week of production time. Scripting is even more critical when you work with several motion capture professionals. Scripts enable you to ensure that all data is handled the same way by all operators, resulting in consistent data.

To be really useful, however, your scripts must be readable, easy to customize, and bulletproof. Follow these suggestions when you create scripts and you will find that you and your associates spend less time looking for bugs and more time producing high quality motion capture animation:

- [Use spacing \(page 51\)](#)
- [Add comments \(page 53\)](#)
- [Use a naming convention \(page 53\)](#)

Use spacing

Using the right spacing and indentation makes your scripts easier to read, easier to use, easier to customize. Spacing does not affect the execution of a script, so you can use as many spaces, line breaks, and tabs as you like.



Tip

You must insert at least one space between keywords and variables.

The following is an example of a poorly spaced script. As you can see, it is difficult to understand the structure of the script, to identify the variables, or to figure out what the script does.

```
if ( ( `getCount $ selectionArray ` ) == 2 ){
vector $source1 = $selectionArray[0].Translation; vector $source2 =
$selectionArray[1].Translation; vector $position = ( $source2 - $source1 );
float $distance = `getLength $position `; print ( "The distance between " + (
string ($selectionArray[0]) ) +" and " + ( string ($selectionArray[1]) ) + " is
" + ( string ( $distance ) ) + "mm ." );
} if ( ( `getCount $ selectionArray ` ) != 2 )
print "Must have exactly 2 nodes selected."; $text = "Default scaling";
```

Here is the same script that is easier to read with improved spacing:

```
if( ( `getCount $selectionArray` ) == 2 )
{
    vector $source1 = $selectionArray[0].Translation;
    vector $source2 = $selectionArray[1].Translation;
    vector $position = ( $source2 - $source1 );
    float $distance = `getLength $position `;

    print ( "The distance between " +
        ( string ($selectionArray[0]) ) +
        " and " + ( string ($selectionArray[1]) ) +
        " is " + ( string ( $distance ) ) + "mm ." );
}
else
{
    print "Must have exactly 2 nodes selected.";
}
$text = "Default scaling";
```

Each command statement is on its own line. The If/Then loop elements are on their own lines. Variables are defined on their own lines. Print statements are structured so you can see at a glance what elements are being assembled. Take time to format scripts whenever you create them. You will find that the investment pays off the very next time you need to use them.

Add comments

HSL scripts without comments are like city maps without street names. They might take you somewhere, but you can not be sure it will be where you want to go.

Add comments to your scripts so it is easy for you and for others to understand what each section of the script does, and what data can be changed. To add a comment to a script, just put two forward slashes at the start of the comment line `//`.

```
// Default locator number is lucky.  
int $locator = 7;  
// Select the head markers and make a rigid body
```

Use a naming convention

If you develop and use some kind of consistent strategy you routinely use for defining variables and naming objects, you will save yourself hundreds, if not thousands, of hours of debugging time. Follow these suggestions to use a convention to name variables and objects logically and concisely:

■ Pick a case

Because the Shogun Post HSL Scripting Language is case sensitive, AUX1, aux1, and Aux1 are three entirely different names. Pick a naming strategy you can remember and live with. For example, you may want to make all object names upper case and all variable names lower case. That way you can quickly recognize whether a variable is an object, a variable, or a Shogun Post command (since those are frequently mixed-case). Whatever strategy you select, stick with it. By default, Shogun Post scripts start with a capital letter and use internal capitalization, that is, each key word within the script name starts with a capital letter, for example: FillGapsRigid. This helps to distinguish Shogun Post scripts from commands.

■ Use descriptive variable names

Pick variable names with meaning. Variables like `x`, `l` and `tr` mean nothing two hours after you write them. Avoid being overly descriptive because long times are hard to type and hard to debug. Consider putting in comments near where variables are defined to identify their use. Knowing that `$rad` is radius and not an angle measurement in rads is useful information.

■ Avoid burying user-defined numbers or parameters

Define variables as close to the top of a script as possible. Make the variable assignments rational and easy to understand. Use the variable you set later in the script, and change its value as required. It is easiest to debug scripts when variables are initially defined at the top of a script with values like:

```
float $headOffset = 150;  
string $errorMessage = "Need to select object first.";
```

Let's say you used the `headOffset` variable at the beginning of the script and used that variable instead of the actual number at each place in your script where it was needed. You could quickly change the value of `headOffset` to increase its accuracy.

■ Bulletproof scripting

When composing HSL scripts, keep all levels of users in mind. Make sure that the HSL script considers user errors and handles these errors gracefully. Think about the errors that your HSL script might encounter. After checking for an error and finding that it is present, have a reasonable contingency action in your HSL script for that error, such as politely informing you of the missing components.

Example:

```
module $selected = `getModules -selected -primary`;
int $num_selected = `getCount $selected`;
string $primaryName;
// This conditional will run if there was
// at least one marker selected.
if ( $num_selected >= 1 ){
select;
select $selected[0];
$primaryName = `getProperty $selected[0] "Name" `;
print ( "Made " + string( $primaryName) + " primary and only selection." );
}
// This statement will be executed if no markers were selected.
if ( $num_selected == 0 ){
print "No markers were selected.";}

```

In this example, if the user has not selected any markers, the script creates an error message rather than failing.

■ Never overwrite data or files automatically

Many scripts modify hundreds or thousands of keys. Making a script which bakes that data over original data or saves a file that overwrites the original is asking for disaster. Obviously sooner or later someone is going to run that script, mangle data, and destroy the good data they had to start with. HSL scripting tools make it easy to save files with new names or create data on clips or layers. Use that functionality to protect your raw data.

Use HSL scripts to work with data

You can use scripts to address and access data in the following ways:

- [Work with variables in HSL scripts \(page 55\)](#)
- [Understand reserved words \(page 60\)](#)
- [Work with HSL data types \(page 65\)](#)
- [Work with HSL operators \(page 75\)](#)
- [Control the flow of HSL scripts \(page 82\)](#)
- [Group HSL statements for scope control \(page 91\)](#)
- [Call HSL scripts \(page 92\)](#)
- [Work with HSL return values \(page 92\)](#)
- [Print HSL command data to Log \(page 94\)](#)
- [Retrieve data using HSL scripts \(page 94\)](#)

Work with variables in HSL scripts

A variable is a name that you use in a command to represent a type of data. The variable is replaced with a specific data value when the command is executed. A variable consists of its name and a data type, which identifies the type of value the variable represents.

A variable is used as an argument in Shogun Post command syntax (see [Understand HSL command syntax \(page 48\)](#)).

The following topics explain how to:

- [Declare and assign variables \(page 56\)](#)
- [Use global variables \(page 59\)](#)

For information on reserved words, also see [Understand reserved words \(page 60\)](#).

Declare and assign variables

Declaring a variable means defining its name and data type. You must declare variables before you can use them. You can declare multiple variables on a single script line. It is wise to declare the variables a script will use near the top of the script, and to provide comments which detail what each variable does.

You declare a variable using the following syntax:

```
data_type $variable_name;
```

where:

data_type	identifies the type of data object that the variable can contain
-----------	--

variable_name	is the name of the variable.
---------------	------------------------------

Assigning a variable means giving a specific value to a declared variable. You assign a value to a variable using the equals sign (=). This is usually done to assign the results of an operation to a variable.

An assignment statement, is written with the variable followed by the equals sign, followed by the value assigned to the variable:

```
data_type $variable_name = value;
```

For example, to assign the type and value assigned to a variable named monster:

```
float $monster = 3.8;
```

The result of this assignment statement is a value of 3.8 being set for a variable named monster with a data type of float. This result can be used as a constant of the same type and value.

Variable data types

This table shows the data types that can be used for Shogun Post variables.

Variable data type	Definition	Examples
Boolean	Boolean flag. See Boolean (page 65) .	true
Float	Floating point number. See Float (page 65) .	265.8, -973.2
Int	Integer. See Int (page 65) .	0, 4, -7
String	Characters. See String (page 66) .	"My mother has a car!"
Vector	One-dimensional array of 3 floating points or integers (stored as floating points). See Vector (page 68) .	<<1, 8.2, 7.6>>

Variable names

When naming a variable, note that the name:

- Must begin with a dollar sign (\$)
- Must not contain spaces between characters
- Must not begin with a number
- Must not use reserved words (see [Understand reserved words \(page 60\)](#))
- Is case sensitive; note whether letters should be upper or lowercase

Variable syntax examples

The following examples show the correct use of variables:

```
string $boneName;
int $_sample_set;
string $name;
```

The following examples show the incorrect use of variables:

```
float $2PI;
vector cheese;
```

The first example is invalid because it starts with number. The second example is invalid because it does not start with the dollar sign (\$).

Declaration syntax examples

The following example shows how to declare a variable of type string called boneName which will be replaced with a bone name.

```
string $boneName;
```

The following example shows how to declare multiple variables:

```
string $str1, $str2, $str3;
```

Assignment syntax examples

The following examples show how to assign specific values to an integer, float, and vector variable respectively:

```
int $frame = 1;
float $distance = 9.54322;
vector $position = <<1, 2.345, 6.789>>;
```

The following example declares and assigns arrays:

```
int $test[2];
$test[0] = 1;
$test[1] = 2;
```

Use global variables

Global variables are variables which persist across scripts and until Shogun Post is closed.

Here is an example of setting a global variable:

```
// Sets a global variable.  
setGlobalVar $myVar value;  
// Gets a global variable.  
// Appropriate command must be used depending on variable type.  
getGlobalFloatVar;  
getGlobalIntVar;  
getGlobalStringVar;
```



Caution

There is no way to delete a global variable, or to list global variables that have been set.

In general, global variables are something you want to use very sparingly. Shogun Post scripts can take parameters (inputs) and return a value, which reduces the need for global variables.

Understand reserved words

Reserved words define control constructs, data types, and Boolean states in Shogun Post scripts. Do not use reserved words as variable names in your script commands.

The following are reserved words.

```
assert
boolean
break
case
continue
default
do
else
false
float
for
function
if
input
int
no
off
on
return
string
switch
toggle
true
vector
void
while
yes
```

See also [input \(HSL reserved word\) \(page 61\)](#).

input (HSL reserved word)

Summary

input is a reserved word. It is used in HSL scripts to define a variable as an argument that is required to run the script.

Details

input can be used to define control constructs, data types, and Boolean states in Shogun Post scripts. Do not use input as a variable name in your script commands.

Inputs are commonly used to create parameters for a script, which are visible in the Pipelines panel. Input statements set the default value of a variable, and some conditions that must be adhered to, such as range. The variable named in the input statement must be declared in the script. The input statement also allows you to control the way in which the parameter is visible and specified in the Pipelines panel.

Command syntax

```
input variableName defaultValue [-advanced] [-allowNonDiscrete] [-color]
[-description string][-discrete value1 value2 ..] [-file Filter string] [-
folder] [-macro listItem1 script1 listItem2 script2] [-module] [-
namedDiscrete listItem1 value1 listItem2 value2 ..] [-range minNumber
maxNumber]
```

Arguments

Name	Type	Required	Comments
variableName	string	Yes	Variable name declared in the script, must begin with \$.
defaultValue	Must match type of variableName	Yes	Default value that the variable will take, if not given as an argument when calling the script.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
advanced	0		–	Makes the parameter for the script advanced, rather than always visible in the Pipelines panel.
allowNonDiscrete	0			Should only be used when the discrete or namedDiscrete flag is present. When using the discrete flag, this allows the input parameter in the Pipelines panel to be edited as well being selected from the discrete picklist
color*	0		file, folder, module, discrete, namedDiscrete	Adds a color picker to the pipeline parameter. The associated variable must be type vector.
description	11	string	–	Adds a description that is visible in the Pipelines panel.
discrete*	Many	string	color, file, folder, module, namedDiscrete	Sets a picklist for the value which is visible in the Pipelines panel.
file*	1	string	color, folder, module, discrete, namedDiscrete	Generates a file icon in the parameter in the Pipeline panel enabling browsing to a file. The fileFilter sets the filter in the file browser. The associated variable must be type string.
folder*	1	string	color, file, module, discrete, namedDiscrete	As with file (above), except string returned is a path to a directory rather than a file.

Name	Flag arguments	Argument type	Exclusive to	Comments
macro	Many pairs	string	–	Enables the input to be returned from a script. Pairs of arguments are specified as strings. The first string is a picklist item visible in the Pipelines panel; the following string is a script name that will return the input value.
module*	0		color, file, folder, discrete, namedDiscrete	Enables you to pick a module from the scene to use in a string variable.
namedDiscrete*	Many pairs	string, variable type	color, file, folder, module, discrete	Enables you to create a picklist of type string, which has corresponding values, the same type as the variable type.
range	2	int or float		Enables you to set a range for integer or float variables. May only be used when the variable type is numeric.

* This flag changes the way the variable's input is entered in the GUI and therefore may not be combined with any of the other flags marked with an asterisk * in the above table. You can use these flags only when the variable type is compatible, for example, you cannot use -folder with an int variable. However, you can use color, discrete, or namedDiscrete for a color variable.

Return value

void

Example

```
// -macro
// This allows you to specify pairs of arguments as strings after the -macro.
// First string is the value in the picklist in the operation parameters
// Second string is the script to run which will return a value into the input variable.
// IMPORTANT all return variables from scripts must be the same variable type,
// in this case strings. Otherwise it might crash Shogun Post.
string $GetDayOrMonthAsString;
input $GetDayOrMonthAsString "" -macro "Day" "GetDayOfWeek" "Month" "GetMonth";
```

```

print $GetDayOrMonthAsString;
// -module
// This allows you to pick a module from the scene to use in a string variable
// If there is a blank scene there will be nothing to pick
// associated variable must be type string
string $PickModuleName;
input $PickModuleName "" -module;
print $PickModuleName;
// -discrete -allowNonDiscrete
// Example 1: allows you to create a picklist using the -discrete switch
// NOTE: The default value for the input has to be included in the list of discrete inputs
string $PickTextToPrint;
input $PickTextToPrint "Text1" -discrete "Text1" "Text2";
print $PickTextToPrint;
// Example 2: allows you to edit the text as well as use the picklist using the
// -allowNonDiscrete switch
// NOTE: It is helpful, but not essential, to add "" to the picklist, so that it is obvious
// you can leave blank or enter text
string $PickOrTypeTextToPrint;
input $PickOrTypeTextToPrint "Text3" -discrete "" "Text3" "Text4" -allowNonDiscrete;
print $PickOrTypeTextToPrint;
// -namedDiscrete
// Specify pairs of strings, first string: Picklist value, second string: Value to set variable to.
// Allows you to create a picklist, that will convert to another value of the same variable type
// Picking "One" will set the variable to "1", which could be easily converted to an integer.
// IMPORTANT: The default value must be a valid entry from the variable value, and not
// the picklist value.
string $GetNumberFromWord;
input $GetNumberFromWord "1" -namedDiscrete "One" "1" "Two" "2" "Three" "3";
print $GetNumberFromWord;
// -advanced
// This will make the parameter for the script an advanced one rather than always visible in
// the pipelines panel.
// -color
// This will add a color picker to the pipeline parameter
// associated variable must be type vector
// -description
// This will add a description which will be visible in the pipelines panel.
// -file
// This generates a file icon in the parameter in the pipeline panel allowing browsing to a path
// associated variable must be type string
string $Filename;
input $Filename "" -file "ShogunPost File (.hdf)|.hdf";
print $Filename;
// -folder
// As with file, except string returned is a path to a directory rather than a file.
// -range
// Allows a range to be set for integer or float variables
float $EnterValInRange;
input $EnterValInRange 10.000 -range -50.000 50.000 -description "Enter value between -50.000 and
50.000";
print $EnterValInRange;

```

Work with HSL data types

The following topics describe the types of data you can use in Shogun Post scripting, and shows how you can use them in arrays and convert a data type to another type.

- [Understand data types \(page 65\)](#)
- [Use arrays \(page 71\)](#)
- [Use data type conversions \(page 72\)](#)

Understand data types

You can address and access the following types of data from within a script:

- [Boolean \(page 65\)](#)
- [Float \(page 65\)](#)
- [Int \(page 65\)](#)
- [String \(page 66\)](#)
- [Vector \(page 68\)](#)

Boolean

A boolean data type is a Boolean (logical) flag, that is one whose values can be evaluated as:

- true / false
- on / off
- yes / no
- toggle state

Float

A float data type is a floating point number, that is one that contains a decimal point.

Int

An int data type is a an integer, that is a whole number.

String

A string data type is any sequence of characters (letters, numbers, or special characters). Character names, marker names, bone names, and all other objects in Shogun Post are addressed using strings. In order to work with those objects, you must declare strings for them.

Objects or modules in Shogun Post are not required to have unique names as long as their paths are unique. In order to reference a particular object in the scene using a string, you must specify its module path. A module path is the path traversed from the top of the Shogun Post object hierarchy down to the module. A path you specify for a string in your script can consist of a full path, partial path, or only the object name.



Tip

You can view an object's path in the **Selection** panel (click the Hierarchy button to organize the objects by hierarchy) or a **VPL** workspace.

Strings do not need to be surrounded in quotation marks unless they contain spaces or start with a number. You can include variables and special characters in strings by using the \ (backslash) escape character and a double quotation mark. You can concatenate (append) strings to other strings using the plus sign + arithmetic operator.

String examples

These examples show you how to declare and work with strings.

```
// Declare a single string
string $markers = "LFHD";

// Append strings
string $string = "test" + "one";

// Include quotation mark
Print "print the \" ";

// Specify full path name
Actor_1\root\lfemur\ltibia

// Look for top-level node named rfemur
rfemur\>

// Find first rfinger object that has no children
// (i.e. is a leaf node in the hierarchy)
<\rfinger

// Find first descendent of relbow named rfinger where relbow is a top-level node
relbow\\rfinger
// Find first descendent of rfemur named rfoot where rfemur may be a
// top-level node
\\rfemur\\rfoot

// Find descendent named ltoe of Actor1 whose immediate parent is lfoot:
Actor_1\\lfoot\ltoe

// Select objects by module path and name
select rfemur\>
zoomView -selected;
```

Strings sample script

This example shows how to declare and work with strings.

```
// Declare your strings
string $a ;
string $b ;
string $c ;
boolean $start_pos;
// Retrieve some properties
$a = `getStringProperty LFHD Rotation_Order` ;
print $a;
$b = getStringProperty( LFHD, Name );
print $b;
// Return array of any selected modules in the scene
string $selected[] = `getModules -selected`;
string $junk = string($selected[0]);
print $junk;
// Determine if that module has any keys on it or not
$start_pos = `hasKey $junk Translation`;
print ( " Does the object have any keys " + string ($start_pos) );
```

Vector

A vector data type is a one-dimensional array of three floating points or integers (stored as floating points). This data type is used to list X,Y,Z positions or X,Y,Z rotations for a given object. When you are working with captured marker data or animated skeletons, you frequently have to edit or evaluate hundreds or thousands of points stored in vector variables.

You assign components of vectors by including a period (.) between the vector variable name and each x, y, or z component. For example:

```
vectorName.x
```

Vector examples

These examples show you how to assign a vector variable and display its x,y, and z values to the Log:

```
//Assign three vector values separately
vector $translation = <<2.33, 4.55, 6.57>>;
float $ Xcomponent = $ translation.x; //Assigned 2.33
float $ Ycomponent = $ translation.y; // Assigned 4.55
float $ Zcomponent = $ translation.z; // Assigned 6.57
print $translation;
//Assign all three vector values simultaneously
vector $rot = <<8.99, 0.11, 2.33>>;
float $ Xcomponent = $ rot.X; //Assigned 8.99
float $ Ycomponent = $ rot.Y; //Assigned 0.11
float $ Zcomponent = $ rot.Z; //Assigned 2.33
```

```
print $rot;
print $ Xcomponent;
print $ Ycomponent;
print $ Zcomponent;
```

This example shows you how to assign a vector variable for rotation:

```
rotation $rotation = <[8.99, 0.11, 2.33]>;
float $Xcomponent = $rotation.x;    //Assigned 8.99
float $Ycomponent = $rotation.y;    //Assigned 0.11
float $Zcomponent = $rotation.z;    //Assigned 2.33
```

The script below can be run on an empty scene. It shows you how to set the value of a vector to any set of three numbers. This is useful because you can use this to set keys by script.

```
// Define the variable you will use to store the vector
vector $vec;
// Assign the vector as a set of three numbers
$vec = <<1,0,0>>;
// Print the $vec variable
print $vec;
// Change the value of the magnitude of a vector's translation value
setLength $vec 2.6;
// print the $vector
print $vec;
```

This example shows you how to get the dot product of two vectors.

```
vector $a = <<5 , 1, 6 >>;
vector $b = <<4 , 4, 2 >>;
float $c = `dot $a $b`;
print (string ($c));
```

Vector sample script

You can use the following script as a guide and a template for working with vector data. Note that this is not the most useful application of vector editing or arithmetic but its an easy to understand application which can be adapted to many purposes. Refer to the comments to see what each set of statements does.



Important

If you want to use this script, you need to import marker data first.

```
// Define variables for use in the script. Note that the $mod
// variable is used to store the name of the object you want to adjust.
// The vector variables, $vec and $rot will store position and rotation values
string $mod = LFWT;
vector $vec = `getVectorProperty $mod Translation`;
vector $rot = `getVectorProperty $mod Rotation`;
// In the following line, you define $output as a statement that
// identifies the object being modified, and its X, Y, Z position
string $output = string( $mod ) + "'s Translation is " + string( $vec );
// This prints the statement.
print( $output );
// This set of statements selects the $mod object, selects its
// entire ranges, then all keys in that range
select $mod;
selectRange -all;
selectKeys -ranges;
// This statement sets the X, Y, Z position of $mod
// to the X, Y and Z values stored in the $vec variable
setProperty -r "Translation" $vec.x $vec.y $vec.z;
print( "Just set the property on LBWT!!! Ugly, ain't it?" );
```

Use arrays

Shogun Post allows you to declare an array of boolean, float, int, string, or vector data types.

You declare (initialize) an array by including the array size in square brackets after the variable name. For example:

```
data_type $variable_name[array_size]
```



Important

Once you have declared an array, you cannot change its size. However, you can use a variable to establish the initial size of the array.

Shogun Post uses a zero-based system, so in order to access an element in an array, the index of the first element must be 0 (zero).

You assign a value to an element of the array using the equals sign (=). For example:

```
$variable_name[array_element] = [value]
```

Array examples

This example shows you how to create an array and print the data to the command line:

```
int $frames[3];
$frames[0] = 1; //Assigned 1 to first element of array
$frames[1] = 5; //Assigned 5 to second element of array
$frames[2] = 10; //Assigned 10 to third element of array
print $frames;
```

This example shows you how to assign a value of 30 to the second element of an array (remember, array elements start at 0 (zero)):

```
$intArray[1] = 30;
```

This example shows you how to create output in sequential order:

```
int $array[3];
$array[0] = 1;
$array[1] = 2;
$array[2] = 3;
print $array;
```

This gives the output:

```
1
2
3
```

However, note that the following example produces output in reverse order:

```
int $array[];
$array = [[ 1, 2, 3 ]];
print $array;
```

This gives the output:

```
3
2
1
```

Use data type conversions

From time to time you will have to convert data from one type to another. For example, you may need to convert an integer to a string.

You can use the following methods to convert data from one type to another:

- [Explicit type conversion \(page 72\)](#)
- [Integer division truncation \(page 73\)](#)
- [Precision and Maximum Numerical Sizes \(page 73\)](#)
- [Range Wrap-Around \(page 74\)](#)

Explicit type conversion

You can convert values to different data types using type casting. This script fragment demonstrates type conversion and prints results to the Log.

```
// Value:15.0
float $conversion = float(15);
int $start = 123;
string $frame = string($start); // Value: "123"
// printing values
print $start;
print $frame;
print $conversion;
```


Integer division truncation

In arithmetic operations, Shogun Post will use displayed values to determine the data type of a constant or variable when no data type is declared. For example:

```
// Result: 0
float $tuna = 1/2;
print $tuna;
```

The numbers 1 and 2 are integers because they have no decimal points. Therefore, integer 1 is divided by integer 2, resulting in 0 with a remainder of 1. Shogun Post discards the remainder.

In this example, tuna is a float variable, so Shogun Post converts the integer value 0 to floating point value 0, then assigns this value to tuna. The fractional component of the value can be obtained by converting one or more of the integer operands to a float. For example:

```
//Result: 0.5
float $tuna = 1.0/2.0;
print $tuna;
```

By adding the decimal point, 2.0 becomes a floating point number. The number 1 is converted to a float and divided by the number 2.0, resulting in the value 0.5 which is assigned to tuna. This may also be accomplished using the method shown in the following example:

```
//Result: 0.5
float $tuna = float (1) /2;
print $tuna;
```

The number 1 is converted to a float, thus causing the 2 to be converted to a float also. This allows a float division to take place, and maintains the fractional result of 0.5.

Precision and maximum numerical sizes

There are limits to the precision sizes of floats and integers. The maximum size is dependent on the user's machine, as defined in the C++ programming language. Floats have limited precision, so you must be careful to avoid round-off errors that occur in long calculations (when exceeding fifteen digits).

```
//Value: -1294967296.000000
float $sushi = 1.5 + 1000000000 3;
print $sushi;
```

Range wrap-around

Exceeding the limited range of a variable causes it to wrap-around to the variable's minimum range which may result in errors. The negative results may occur when the minimum range of a variable is exceeded and it wraps around to the variable's maximum range.

```
//Value: -2147483648
int $tuna = 2147483647 + 1;
//Value: 2147483647
int $salmon = -2147483648 - 1;
//Value: -2147483648
int $dolphin = 2147483648;
//Value: -2147483647
int $mackerel = 2147483647 +2;
print $tuna;
print $salmon;
print $dolphin;
print $mackerel;
```

Work with HSL operators

Operators enable you to manipulate data with commands. An operator is a symbol that represents an action to be taken by the command. The operator acts on an operand, the object of the action. In Shogun Post scripting, an operand can be a variable or a constant value.

Operators require operands to be of the same type. If there is any difference between the left and right variable, one may be converted to the value of the other.

Operators are written as part of an expression, with the operator between the operands in the following format:

```
operand_1 operator operand_2
```

For example:

```
int $frame = 1;
```

where:

<code>int \$frame</code>	is the left side operand, in this case a variable.
--------------------------	--

<code>=</code>	is the operator, in this case an assignment operator
----------------	--

<code>1;</code>	is the right side operand, in this case the value assigned to the variable
-----------------	--

The following sections show you how to work with operators:

- [Understand operator types \(page 75\)](#)
- [Determine operator precedence \(page 81\)](#)

Understand operator types

Shogun Post scripting recognizes the following types of operators:

- [Arithmetic operators \(page 75\)](#)
- [Comparison operators \(page 78\)](#)
- [Logical operators \(page 79\)](#)

Arithmetic operators

Arithmetic operators are symbols that represent mathematic functions.

An expression with an arithmetic operator, also called an arithmetic statement, the arithmetic operator is written with the first operand followed by a mathematic symbol, followed by the second operand:

```
operand_1 mathematizations operand_2
```

This table shows the symbols that Shogun Post recognizes as arithmetic operators, defines their function, and indicates the data types that can be used as operands.

Symbol	Function	Valid data types
+	addition	float, int, string, vector, rotation
-	subtraction, negation	float, int, string, vector, rotation
*	multiplication (int, float) or dot product (vector)	float, int, vector
/	division	float, int, vector
%	modulus (remainder of division)	float, int
^	power (int, float) or cross product (vector)	float, int, vector

Arithmetic operator shortcuts

Shortcut operations enable you to assign a variable the value of itself operated on by some other value. You can write shortcut expressions for any arithmetic operators.

For example, instead of writing the full expression in the format:

```
variable = variable operator value
```

you can write the following shortcut format:

```
variable operator = value
```

The following table shows all of the shortcut operators of this form, their expanded syntax, and their values.

Shortcut syntax	Expanded syntax	Value
variable += value;	variable = variable + value;	variable + value;
variable -= value;	variable = variable - value;	variable - value;
variable *= value;	variable = variable value;	variable value;
variable /= value;	variable = variable / value;	variable / value;
variable %= value;	variable = variable % value;	variable % value;
variable ^= value;	variable = variable ^ value;	variable ^ value

You can also write shortcut expressions to add and assign int, float, and vector data types at the same time.

For example, instead of writing the full expression:

```
$hammer = $hammer + 3.5;
```

You can write the following shortcut:

```
$hammer += 3.5;
```

Both statements accomplish the same results.

Arithmetic operator examples

These examples show you how to write expressions using arithmetic operators for the specified data types.

Integers and floats

Basic math rules apply for integer, floating-point attributes, and variables. The modulus (%) operator calculates the remainder of division. For example,

```
int $plop = 11 % 3; //Value: 2
```

Strings

Strings can be concatenated, or appended to other strings, with the plus sign + operator. For example:

```
// Starts out empty
string $words = "Thanks for all" + " the fish";
string $myStr;
// Value: " from the market"
$myStr += " from the market";

// Value "Thanks for all the fish from the market"
string $fullStr = $words + $myStr;
```

Vectors

The power (^) operator calculates the cross product of vectors, and the multiplication * operator calculates the dot product of vectors. For all other operators, each component of one vector is operated on by its counterpart component in the other vector.

```
//Value: -4.000000, 2.000000, 0.000000
vector $test = <<1, 2, 3>> ^ <<2, 4, 4>>;
print ($test);
```

Operator shortcut examples

These examples show how to use shortcut operators rather than writing out full expressions:

```
// Declare variable
int $inside = 1;

// Shortcut for expression $inside = $inside + 9
$inside += 9;

// Shortcut for expression $inside = $inside -2
$inside -= 2;
```

```
// Shortcut for expression $inside = $inside 10
$inside *= 10;
print $inside;

//declare variable
vector $outside = <<2,2,2>>;

// Shortcut for expression $outside += $outside + <<<1,2,3>>
$outside += <<1,2,3>>;
print $outside;
```

Comparison operators

Comparison operators are used to compare two values and evaluate whether a statement is true based on the relation of the value on the left to the value on the right and then execute the statement accordingly.

Comparison operators are valid with int, float, and vector data types. The truth of the statement depends on the relation of the value on the left to the value on the right of the operator.

An expression with an comparison operator, also called a comparison statement, is written with the first operand followed by a relational symbol, followed by the second operand:

```
operand_1 relational_symbol operand_2
```

This table shows the symbols that Shogun Post recognizes as relational operators and indicates the condition that would result in the statement being evaluated as true.

Symbol	True only if the left-hand side is
<	less than the right-hand side
>	greater than the right-hand side
==	equal to the right-hand side
!=	not equal to the right-hand side
>=	greater than or equal to the right-hand side
<=	less than or equal to the right-hand side

Note that comparison operators and arithmetic operators can be used in the same statement.

```
// Prints "I love you"
float $threshold = 10.0;
if( $threshold < 25.0 )
print( "I love you" );
```

You can write a hook statement with a comparison operator using three operands.

Examples of comparison expressions

```
//true
if (3.5 < 6) print ("true");
//false
if (10 < 10) print ("true");
//false
if (56 == 57) print ("true");
//false
if (-3 != 3) print ("true");
//true
if (-3 >= -3) print ("true");
//false
if (0 <= -3) print ("true");
```

Hook statements

A hook statement is written with three operands. The first operand is a test condition that represents true or false. This operand is followed by a question mark ?, then the second and third operands are separated by a colon (:).

```
operand_1 ? operand_2 : operand_3
```

If the test condition is true, the operand after the question mark ?, is used. Otherwise the operand after the colon (:) is used. For example:

```
test condition ? operation1 : operation2
int $markers = 5 < 9 ? 40 : 50; // Value: 40
```

Logical operators

Logical operators are used to compare two values and evaluate whether a statement is true or false and then execute the statement accordingly. Logical operators create boolean results, that is, values of either TRUE or FALSE.

You can use logical operators with int, float, and vector data types. The values of each type correspond to being either true or false:

- An int or float value is considered false when its value, converted to type int, is 0.
- A vector value is considered to be false when its magnitude, converted to type int, is 0.
- All other converted values are considered to be true.

An expression with a logical operator, also called a logical statement, is written with the first operand followed by a logic symbol, followed by the second operand:

```
operand_1 logic_symbol operand_2
```

This table shows the symbols that Shogun Post recognizes as logical operators, defines their function, and indicates how the statement is evaluated as true.

Symbol	Logic	True only if
	OR	either left-hand or right-hand side is true
&&	AND	both left-hand and right-hand sides are true
!	NOT	right-hand side is false (not supported yet)

Examples of logical operators

```
//true
if ( true || false) print ("true");
//false
if ( true && false) print ("true");
//false
if (no || no) print ("true");
//false
if (off || off) print ("true");
```


Determine operator precedence

If you use multiple operators on the same row, they have equal precedence. If a statement has two or more operators with the same precedence, the left-most operator is evaluated first. Unary and assignment operators (=) are right associative; all others are left associative.

The following sections list operator precedence from highest to lowest and show you how to influence precedence by grouping operators.

Highest precedence

```
( ) [ ]
* / % ^
+ -
< <= > >=
== !=
&&
||
?:
= += -= *= /=
```

Lowest precedence

Examples of when precedence effects the value of a statement:

```
// $front = (5 +(2*3))
int $front = 5 + 2 3

// $side = $side ((8 % 5) / 3)
int $back = 2 - 3 int (1 < 3);

// $side = $side ((8 % 5) / 3)
int $side = 2;
$side = 8 % 5 / 3;
```

Grouping operations for precedence

The order of operation can be manually dictated by placing parentheses around operators. This is because parentheses have highest operator precedence. Examples of dictating precedence:

```
// $side = $side (8 % (5 / 3))
int $side = 2;
$side *= 8 % (5 / 3);
int $top = (5 ( int($side >4)));
```

Control the flow of HSL scripts

You can control the flow of script execution by using the following types of statements to manage the order in which operations are executed:

- [Use conditional statements \(if, else, else if, switch\) \(page 82\)](#)
- [Use looping statements \(while, do while, for\) \(page 86\)](#)
- [Use interruption statements \(continue, break\) \(page 89\)](#)

Use conditional statements (if, else, else if, switch)

Conditional statements execute a command (or grouping of commands) when a specific test condition is true. This flow control mechanism is useful if you have created a series of commands that will be executed only if a certain condition exists.

You write a test condition by enclosed it in parentheses (). The resulting value must be an int, float, or vector data type. The test condition value is then converted to an int data type to represent either true or false.

```
// if (test condition) execute statement;  
if (0 < 1) print "zebra";
```

If the test condition has a value of type int or float, the test condition is true only if the value, converted to type int is not 0:

```
if (<<0,0,0>> == <<0,0,0>>) print "referee"; //true
```

You can use the following types of conditional statements in Shogun Post:

- [IF statements \(page 83\)](#)
- [ELSE statements \(page 83\)](#)
- [ELSE IF statements \(page 84\)](#)
- [SWITCH statements \(page 85\)](#)



Tip

To make conditional statements easy to debug, group the commands to be executed under the IF statement.

IF statements

The IF statement works as follows: if the test condition is true, the statement is executed.

You write an IF statement expression in this format:

```
if (test condition)
  execute statement;
```

If you are using multiple statements, use grouping to control scope execution by enclosing the statements within braces {}.

For example:

```
if ( 2 < 3 )
{
  string $blah = "This worked";
  print ($blah);
}
```

ELSE statements

The ELSE statement always follows the IF statement. The ELSE statement is executed when the preceding IF statement is false.

You write an ELSE statement expression in this format:

```
if (test condition)
{
  execute statement_1;
}
else
{
  execute statement_2;
}
```

For example:

```
if (5 % 2 > 1)
{
  print "bah";
}
else
{
  print "humbug";
}
```

ELSE IF statements

The ELSE IF statement is always paired with the IF conditional statement. Unlike the else command which executes a statement when the IF statement is false, the ELSE IF statement requires another test condition to be true, or it will not execute either statement.

You write an ELSE IF statement expression in this format:

```
if (test_condition_A)
{
    execute_statement_1;
}
else if (test_condition_B)
{
    execute_statement_2;
}
```

For example:

```
if (-1 >= 1)
{
    print "Statement 1";
}
else if (-1 <= 1)
{
    print "Statement 2";
}
```

SWITCH statements

The switch statement is a conditional statement that executes a code block depending on which one of a number of given cases is matched to the switch expression.

The switch statement takes either a float, int, Boolean, or string as an argument. Each case must be the same data type as the switch expression, that is, if the switch expression takes an int, all cases within the switch statement must also be int, as shown in the following example.

The optional keyword `break` is often used in switch statements to break out of the statement when one of the cases is found to match the expression, as shown below.

You write a switch statement in this format:

```
switch (expression)
{
  case n:
    code block
    break;
  case n:
    code block
    break;
}
```

For example:

```
int $test = 2;
switch($test)
{
  case 1:
    print "1";
    break;
  case 2:
    print "2";
    break;
  case 3:
    print "3";
    break;
}
```

Use looping statements (while, do while, for)

Looping statements repeatedly execute statements while a specific test condition is true. This flow control mechanism is useful when you have a set of statements you want to run until a certain condition is met.

You can nest looping statements.

You can use the following types of looping statements in Shogun Post:

- [WHILE statements \(page 86\)](#)
- [DO WHILE statements \(page 87\)](#)
- [FOR statements \(page 87\)](#)



Tip

You can use Interruption statements to affect the flow control in looping statements.

WHILE statements

A WHILE looping statement continuously executes designated statements as long as a specific condition remains true.

You write a WHILE looping statement in this format:

```
while (test condition)
execute statement;
```

For example:

```
int $PJ = 5;
while ($PJ > 0)
{
    print
    ("There are " + string($PJ) + " P.J. sandwiches left.");
    $PJ = $PJ - 1;
}
print ("Houston, we have a problem!");
```

DO WHILE statements

A DO WHILE looping statement executes a specific statement then checks whether or not to repeat the loop. This ensures that the statement is executed at least once.

You write a DO WHILE looping statement in this format:

```
do
  execute statement;
while (test condition);
```

For example:

```
int $peanuts = 1;
do {
  print ("Elephants work for peanuts");
  $peanuts += 1;
  print ($peanuts);
}
while ( $peanuts < 10);
```

FOR statements

A FOR looping statement provides loop control that has initialization, test, and an increment statement.

You write a FOR looping statement in this format:

```
for (initializers; test condition; incrementor)
  execute statement;
```

For example:

```
string $blah;
int $i;
for( $i = 1; $i < 6; $i += 1 )
{
  $blah = "This is what happens when things coincide";
  print( $blah );
}
```

Looping examples

These examples show you how to use looping statements.

Single and nested loops

You can use the script below as a template for creating single and nested loops. Refer to the comments to learn what each set of statements does. You may want to save this script to a file called loop-example.hsl for future reference.

```
//Define the statement you are going to print each iteration
string $statement;
//Define variables to use as counters
int $i, $ j;
//Start the $i loop and specify start, end, and amount to increment
for( $i = 1; $i < 6; $i += 1 ) {
//Start the $ j loop and specify start, end, and amount to increment
for( $j = 1; $j < 15; $j += 2 ) {
//Tell Shogun Post to print something if $i < $j
if( $i < $j )
    $statement = string( $i ) + " is less than " + string( $j );
//Tell Shogun Post to print something else if $i > $j
else if( $i > $j )
    $statement = string( $i ) + " is greater than " + string( $j );
//Tell Shogun Post what to print if the two counters are equal
else
    $statement = "This is what happens when things coincide";
//Tell Shogun Post to actually print the statement you've assembled
print( $statement );
//Close the second loop
}
//Close the first loop
}
```

Loop through objects in a scene

Sometimes you may need to cycle through all the objects in a scene in order to find an object you cannot identify, initially, by name. On other occasions you may wish to create a list of all the objects in a scene, or a list of all the new objects added to a scene.

The following example shows you how to list all the objects in a scene by name. You can modify this set of steps to perform the tasks listed above, and to handle any activity which requires you to execute one or more operations on through a given set of objects in a scene.

```
// Courtesy of Jake Wilson of SCEA
// Initialize Variables
print " ";
// Create array of children under 'Actor' Mary's character node
string $childarray[] = `getChildren Mary`;
// Determine # of children
```



```
int $childcount = `getCount $childarray`;
// Create a $childarrayloop to hold a number
int $childarrayloop = 0;
// Print an empty string
string $ proptest;
// Start a loop which run once for each of Mary's children
for ( $childarrayloop=0 ;
    $childarrayloop < $childcount ;
    $childarrayloop +=1 ){
    // print name of current child marker in array
    print ( string ( $ childarray[$childarrayloop] ) );
    // return the name of the marker and compare the name
    // of the marker to the name"LFWT"
    $ proptest = ` getStringProperty $ childarray[$childarrayloop] Name`;
    if ($ proptest==LKNE){
        // if that LFWT marker is found, this tells the loop to quit
        $ childarrayloop=$childcount;
        print ( "EXITING LOOP" );
    }
}
```

Use interruption statements (continue, break)

You can use interruption statements to affect the flow control in Looping statements.

You can use the following types of interruption statements in Shogun Post:

- [Continue statements \(page 90\)](#)
- [Break statements \(page 90\)](#)

Continue statements

The continue statement is an interruption statement that causes the next iteration of the loop to occur, jumping over any remaining statements in the loop. It works inside all loops.

You write a continue statement in this format:

```
looping_statement (test condition)
continue;
execute statement;
```

For example:

```
int $Time;
for ($Time = 10; $Time > 0 ; $Time -= 1)
{
    print ("Time equals: " + string($Time));
    if ($Time < 2)
        continue;
    print ("Keep Going!");
}
```

Break statements

The BREAK statement is a loop interruption that exits the loop immediately. It works inside all loops.

You write a BREAK statement in this format:

```
looping_statement (test condition)
execute statement;
break;
```

For example:

```
int $parole = 3;
while ($parole < 8)
{
    print (string($parole) + " days until $parole");
    $parole -= 1;
    if ($parole <= 0)
        break;
}
print "monopoly";
```

Group HSL statements for scope control

You can control the scope of script execution by grouping statements into a block. The contents of a group cannot be referenced outside of the block.

Grouping commands is useful if you want to:

- Use a single statement to execute a group of statements in the desired order
- Create script modularity
- Define the visibility of local variables

You group statements by declaring a block within braces (`{}`):

```
if (test condition)
{
    statement_1;
    statement_2;
}
```

For example:

```
if ( 2 < 3 )
{
    string $blah = "This worked";
    print ($blah);
}
```

Grouping examples

This example groups the statement to be executed for an IF conditional statement:

```
float $MasterControl = 1;
float $MarkerControl = 1;
if ($MasterControl == 1 || $MarkerControl == 1)
{
    select LFWT;
    findTail 2;
    cutKeys;
    print ("Operation was run.");
}
```

Call HSL scripts

You can call any script from any other script. For instance, if you can run a script called `fixSword`. `hsl` in your current project by typing `fixSword`; at the command line, then you can also insert or nest this script in any other script. For example:

```
select LFHD LBHD RFHD RBHD;  
fillGaps -rigid -all;  
// nested script follows  
fixSword;  
rewind;  
findGap;
```

You do not have to use the HSL extension when calling a script. However, the called script does have to be located in a directory specified in **Script directories** in the **Shogun Post Preferences** dialog box.

Work with HSL return values

In order to perform many Shogun Post scripting operations or to create custom Shogun Post user windows, you must know how to work with values returned by commands:

- [Use left-hand single quotation marks \(page 92\)](#)
- [Use C calling convention \(page 93\)](#)

Use left-hand single quotation marks

One way to return the value of a command is to enclose a command within left single quotation marks. You can then assign the enclosed command to a variable and display the result in the Log.



Tip

To create a left single quotation mark (```), press the key to the left of the number 1 on your keyboard.

For example:

```
// Define a variable called $a
string $a;
// Set $a to the value returned by the get property command
$a = `getProperty Name LFHD`;
// Print the return value to the Log.
print( $a );
```

This script prints the contents of \$a, LFHD in this case, to the Log. In the example above, the `getProperty` command returns a string. Therefore, the variable on the left side of the equal sign must also be a string to accept this return value.

In the following example, left single quotation marks are used to define a user window:

```
int $dlg;
int $textBox;
// Create the user window. Call it "Test User Window"
$dlg = `createWindow "Test"`;
// Create the Text Box User Control, adding it to User Window we just created
$textBox = `createTextBox $dlg`;
```

Use C calling convention

As an alternative to using left-hand single quotation marks, you can also return values from Shogun Post commands using a C function syntax in order to collect a return value:

```
$a = getProperty( Name, LFHD );
```

The only time this calling convention may not be used is when options are being specified with the function call. The command below could not be written using the C function syntax because it specifies a command option `-all`.

```
select -all;
```

The three following examples are all equivalent statements.

```
snapToSystem LFHD 3.4 5.6 6.7;
snapToSystem( LFHD, 3.4, 5.6, 6.7 );
LFHD.snapToSystem( 3.4, 5.6, 6.7 );
```

Print HSL command data to Log

You can use the print command to write the contents of specified strings to the Log.

The following code fragments review techniques for displaying information collected by a script. Review the comments to learn what each set of codes do. Cut, paste and modify the fragments to create your own scripts. To test most of these fragments you will have to have previously loaded a scene with markers.

```
// The following statement creates an array, or list,  
// of all the MarkerNodes in the scene  
string $modArray[];  
$modArray = `getModules -type "Marker"`;  
  
//This calculates and prints the translation value for  
// RFHD at the current time  
vector $pos = RFHD.getVectorProperty( "Translation" );  
print $pos;  
  
//This returns and prints the elements of a  
// string array  
string $a;  
$a = getStringProperty( $modArray[0], Name );  
print $a;  
  
// This prints a module element's name just by  
// casting the module to a string  
print( string( $modArray[0] ) );  
  
//You can print the value of any object's translation  
// property using the following statement. Modify this  
// statement to print the value of any property.  
print( string( `getVectorProperty $modArray[0] "Translation"` ) );
```

Retrieve data using HSL scripts


The following topics explain how to retrieve data with scripts:

- [Work with ranges \(page 95\)](#)
- [Get the length of a bone \(page 96\)](#)
- [Determine script execution time \(page 96\)](#)
- [Determine which executable file is running \(page 97\)](#)

Work with ranges

You can write a script to retrieve animation and scene ranges. This is useful when you want to limit an editing operation to only the active range or when you want to make sure you apply and operation to all the frames in a scene.

Import the required Shogun file. Set a playRange using the playback bar to start, say, at frame 50 and end at frame 65. Select and copy one of the examples below, then paste it into the Shogun

Post Script Editor. To execute the script, click the Run button .

```
// Assign the variable $ animStartFrame to the animation start frame
// as retrieved by the command getAnimStart;
int $animStartFrame;
$animStartFrame = `getAnimStart`;

// Print the animation start frame as part of a sentence
print ( "The animation start frame is frame " + string ($animStartFrame ) );

// Create a variable called $animEndFrame and assign it the
// value of the animation end frame as retrieved by the
// command getAnimEnd;
int $animEndFrame;
$animEndFrame = `getAnimEnd`;
print( "The animation end frame is frame " + string( $animEndFrame ) );

// Identifies the incoming scene playRange and animation Range
playRange -save;

// This statement just prints the result of getPlayStart
// getPlayEnd as part of a statement
print( "The playRange is " + string (`getPlayStart`) + " to " + string
(`getPlayEnd`) );
```

Get the length of a bone

You can write a script to calculate and display the length of a bone. This is handy when you need a script to fit custom skeletons to markers or scale skeletons up and down.

```
// getLength just tells you what the length of any vector is.
// It's not a bone-specific command though it's useful for working
// with bones.
// Get selected bone
string $bones Array[] = `getModules -selected `;

// Set the offset of that bone as a vector
vector $source = `getVectorProperty $bonesArray[0] Translation`;

// Translate that vector into a float value
float $length = `getLength $source `;
print (strongylosis));
```

Determine script execution time

The following code allows you to time how long a script takes to execute. From time to time you may need to test that a script won't take hours or days to run, or you may want to create "benchmarks" to see which version of a script runs faster, or which machine in a network is more efficient for editing.

Before you execute this script you must load a suitable Shogun file.

```
//Script timer-----
int $sysTimeStart = `getSystemTime`;
int $sysTimeEnd;
string $sysTimeString = (`formatTime $sysTimeStart`);
print ("Script started on " + $sysTimeString);
//Script timer-----
// Insert a nested script or specific script commands below
selectByType Marker;
selectProperty Translation;
selectRange -all;
fillGaps -all;
filter 0.1 35;
//Script timer End-----
print ("Script started " + $sysTimeString);
$sysTimeEnd = `getSystemTime`;
$sysTimeString = (`formatTime $sysTimeEnd`);
print ("Script ended " + $sysTimeString);
//Script timer End-----
```


Determine which executable file is running

The following code shows you how to return the name of the executable file you are running (ShogunPost.exe). You can then pass the results to another script that requires it.

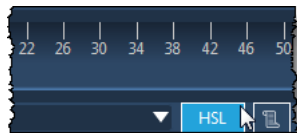
```
string $str = `appInfo "exepath"`;  
string $name = `getFileTitle( str )`;
```

Run scripts from the command line

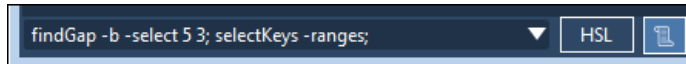
You can enter Shogun Post script commands and view their results in the command line at the bottom of the Shogun Post window.

To run scripts from the Shogun Post command line:

1. Ensure that the command line specifies the scripting language you want to use (HSL or Python). If it does not, to switch the command line to the required language, click the button at the bottom of the Shogun Post window.




2. Type one or more commands in the command line text box. You don't need a semicolon (;) at the end of a single command name. If you want to run more than one command at a time, separate multiple commands with semicolons (;).



3. Press Enter.
4. Check the Log field and the Log button on the status bar for any script execution errors or return values.



Note the following tips for using the command line:

- You can view complete results of all operations executed in the current session of Shogun Post, including scripts you have run from the command line, in the Log window.
- To see all the commands you have previously executed in the current session, click the Log button  to the right of the Log field.
- To execute a command again, click the command.
- To copy a command, select and right-click on text in the command line. You can copy and paste HSL commands from the command line into the Script Editor.

HSL command reference

You can look up all of Shogun's script commands in either of the following ways:

- [Commands in alphabetical order \(page 100\)](#)
- [Commands by category \(page 1383\)](#)

The following information is provided for each Shogun command, where applicable:

- **Description**
 - **Summary:** A brief explanation of the command's function
 - **Details:** A more detailed explanation of the command's function
 - **Functional area:** The category the command belongs to
- **Command syntax**
 - **Syntax:** The syntax for writing the command
 - **Arguments:** Any arguments that can be passed to the command
 - **Flags:** Any flags affecting the command
 - **Return value:** Any values the command returns
 - **Example:** Snippets of sample code illustrating the command's use
- **Additional information**
 - **Related commands:** A list of commands related to this one
 - **Related scripts:** A list of supplied scripts using this command

For more information, see:

- [Understand HSL command syntax \(page 48\)](#)
- [About HSL scripting with Shogun Post \(page 31\)](#)

Commands in alphabetical order

This section lists Shogun HSL scripting commands in alphabetical order.
For lists of commands ordered by category, see [Commands by category \(page 1383\)](#).

abs

Description

Summary

Returns the integer absolute value of a number.

Details

The abs command returns the absolute value of an integer. This return value is always greater than or equal to zero. If a variable of type float is passed in as an argument, the abs command performs a truncating integer conversion, then returns the absolute value of that integer. If the input data is floating point and you want the output type to be floating point, use the [fabs \(page 372\)](#) command.

Functional area

Math

Command syntax

Syntax

```
abs number
```

Arguments

Name	Type	Required	Comments
number	integer	yes	If num is of type float, a truncation is used to convert to int.

Flags

None

Return value

integer

Returns an integer that is greater than or equal to 0

Examples

```
// the following script shows that positive 1 is returned for
// both calls to abs
int $integer_variable = 1.0;
float $float_variable = 1.9;
int $abs_of_float_variable = `abs $float_variable`;
int $abs_of_integer_variable = `abs $integer_variable`;
print("i is " + string($abs_of_integer_variable));
print("f is " + string($abs_of_float_variable));
```

Additional information

Related commands

[fabs \(page 372\)](#)

acos

Description

Summary

Returns the arc cosine (aka inverse cosine) of an input

Details

Arc cosine is the inverse function for cosine. That is, given a value for `val` in the domain $[-1.0, 1.0]$, it returns an angle `acos(val)` in degrees.

To illustrate, picture the unit circle (a circle of radius 1.0 centered about the origin). If you imagine a unit vector rotating inside of the unit circle, the x-coordinate of the endpoint of the vector is the `acos` function argument `val`. The angle that the vector makes with the positive x-axis is the function output `acos(val)`.

Because, as the vector rotates inside the circle, the x-coordinate of the vector must always be within the unit circle, the input argument to `acos` (aka `val`) must always be within $[-1.0, 1.0]$. Also, as $\cos(t)$ is doubly valued for t in $[0, 360]$, the `acos` function is restricted to the first two quadrants. This means that the unit vector rotating inside the unit circle is assumed to have a positive or 0 y-coordinate, therefore `acos(val)` always returns a value in $[0.0, 180.0]$.

Functional area

Math

Command syntax

Syntax

```
acos value
```

Arguments

Name	Type	Required	Comments
value	float	yes	The domain of arc cosine is $[-1.0, 1.0]$. Values of <code>val</code> outside of this domain are not meaningful to the function <code>acos</code> .

Flags

None

Return value

float

For an input argument `val` in `[-1.0, 1.0]`, returns a floating point angle (degrees) in the range `[0.0, 180.0]`.

For input arguments `val` in the ranges: `(1.0, Inf]`, `[-Inf, -1.0)` and `NaN`, `acos(val)` returns `NaN`.

Examples

```
float $t;
float $acos_of_t;
float $cos_of_acos_of_t;
// acos(0) is 90 degrees because cos(90 degrees) = 0
$t              = 0.0;
$acos_of_t      = acos($t);
$cos_of_acos_of_t = cos($acos_of_t);
print("t, acos(t), cos(acos(t)) = " + string($t) + ", " + string($acos_of_t) +
      ", " + string($cos_of_acos_of_t));
// 1.1 is not a meaningful input into acos and returns NaN
// (-1.#IND00)
$t              = 1.1;
$acos_of_t      = acos($t);
$cos_of_acos_of_t = cos($acos_of_t);
print("t, acos(t), cos(acos(t)) = " + string($t) + ", " + string($acos_of_t) +
      ", " + string($cos_of_acos_of_t));
```

Additional info

Related commands

- [asin \(page 144\)](#)
- [atan \(page 148\)](#)
- [cos \(page 230\)](#)
- [sin \(page 1199\)](#)
- [tan \(page 1308\)](#)

addDropListItem

Description

Summary

Adds a string to the user drop list.

Details

Adds a string specified by `itemString` to the drop list user control specified by `userControlID`. Strings are added to the end of the drop list, unless the `-insert` flag is provided, in which case, the string is inserted at the index specified. If the `-sort` flag is provided with the [createDropList \(page 247\)](#) command, the strings in the drop list appear in alphabetical order, and the `-insert` flag has no visible effect.

Functional area

User Window

Command syntax

Syntax

```
addDropListItem userControlID "itemString"[-insert integer]
```

Arguments

Name	Type	Required	Comments
<code>itemString</code>	string	yes	String to add to the drop list.
<code>userControlID</code>	int	yes	ID of user control to operate on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
insert	1	integer	—	—

Return value

integer

Examples

```
// Add items to a Drop List User Control.
int $windowId;
int $controlId;
// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window. Don't use the
// sort flag so we can demonstrate how to insert vs. append.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List. Insert one as well
addDropListItem $controlId "Item 1";
addDropListItem $controlId "Item 2";
addDropListItem $controlId "Item 3";
addDropListItem $controlId "Inserted!" -insert 1;

// Set the current selection to the first item
selectDropListItem $controlId 0;
```

Additional info

Related commands

- [createDropList \(page 247\)](#)
- [deleteAllDropListItems \(page 324\)](#)
- [deleteDropListItem \(page 336\)](#)
- [findDropListItem \(page 401\)](#)
- [getDropListItem \(page 510\)](#)
- [getDropListSelectedItem \(page 513\)](#)

-
- [getNumDropListItems \(page 612\)](#)
 - [selectDropListItem \(page 982\)](#)
 - [setDropListHandler \(page 1058\)](#)

addLayer

Description

Summary

Create a new layer in the active clip

Details

An error is generated if there is no clip in the scene.

Functional area

NLE

Command syntax

Syntax

```
addLayer "layerName"
```

Arguments

Name	Type	Required	Comments
layerName	string	yes	The name of the layer to add

Flags

None

Return value

void

Examples

```
// Print the name of the active clip
string $clip = `getActiveClip`;
print $clip;

// Add new layer
string $name = "new layer";
addLayer $name;
// Print all layers
string $layers [];
$layers = `getLayers`;
print $layers;
```

Additional info

Related commands

- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

addListBoxItem

Description

Summary

Adds a string to the specified user list box.

Details

Adds a string that is specified by `itemString` to the list box user control that is specified by `userControlID`. Strings are added to the end of the list box, unless the `-insert` flag was provided, in which case, the string is inserted at the index specified. If the `-sort` flag was provided with the [createListBox \(page 259\)](#) command, the strings in the list box will appear in alphabetical order, and the `-insert` flag has no visible effect.

Functional area

User Window

Command syntax

Syntax

```
addListBoxItem userControlID "itemString" [-insert integer]
```

Arguments

Name	Type	Required	Comments
<code>itemString</code>	string	yes	String to add to the list box.
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
insert	1	integer	—	Specifies the zero based index which the string should be inserted. If this flag isn't specified, the item will be appended to the end of the list.

Return value

integer

Examples

```
// Add items to a List Box User Control.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window. Don't use the
// sort flag so we can demonstrate how to insert vs. append.
$controlId = `createListBox $windowId`;

// Add some items to the List Box. Insert one as well
addListBoxItem $controlId "Item 1";
addListBoxItem $controlId "Item 2";
addListBoxItem $controlId "Item 3";
addListBoxItem $controlId "Inserted!" -insert 1;

// Set the current selection to the first item
selectListBoxItem $controlId 0;
```

Additional information

Related commands

- [createListBox \(page 259\)](#)
- [deleteAllListBoxItems \(page 326\)](#)

-
- [deleteListBoxItem \(page 340\)](#)
 - [findListBoxItem \(page 407\)](#)
 - [getListBoxItem \(page 578\)](#)
 - [getListBoxSellItems \(page 581\)](#)
 - [getNumListBoxItems \(page 616\)](#)
 - [selectListBoxItem \(page 989\)](#)
 - [setListBoxHandler \(page 1088\)](#)

addListViewItem

Description

Summary

Adds an item to a list view user control.

Details

Adds a string specified by `itemString` to the list view user control specified by `userControlID`. Strings are added to the end of the list view, unless the `-insert` flag is provided, in which case, the string is inserted at the index specified.

For multi-column list views, this command sets the value of the first column. To set subsequent column values, call the [setListViewItemText \(page 1100\)](#) command.

Functional area

User Window

Command syntax

Syntax

```
addListViewItem userControlID "itemString"[-insert integer]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	integer	yes	ID of User Control to operate on.
<code>itemString</code>	string	yes	String to add to the List View.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
insert	1	integer	—	Specifies the zero based index which the string should be inserted. If this flag isn't specified, the item will be appended to the end of the list.

Return value

integer

Returns the index that the item was inserted at.

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columnsstring $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;
```

```
// Add some items to the list view. The text we supply is for the
// first column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- [createListView \(page 262\)](#)
- [deleteAllListViewItems \(page 328\)](#)
- [deleteListViewItem \(page 342\)](#)
- [getListViewItemCheck \(page 583\)](#)
- [getListViewSelItems \(page 589\)](#)
- [getNumListViewItems \(page 620\)](#)

-
- | [selectListViewItem \(page 991\)](#)
 - | [setListViewColumns \(page 1091\)](#)
 - | [setListViewHandler \(page 1094\)](#)
 - | [setListViewItemCheck \(page 1097\)](#)
 - | [setListViewItemText \(page 1100\)](#)

addNamespace

Description

Summary

Adds a selected or specified namespace.

Details

All modules under the same parent can be considered in the namespace of that parent, using slashes (/) to separate each module's parent's name.

addNamespace operates on all selected modules unless you use the `-onMod` flag to specify the module on which to operate by name. If the module already contains a namespace, another is appended to it.

Note that you may specify the path in commands that require a module name, but you may not specify the path as a module name in the Name attribute.

You need to specify only what is required to uniquely identify the name.

Functional area

Namespace

Command syntax

Syntax

```
addNamespace namespace[-onMod string]
```

Arguments

Name	Type	Required	Comments
namespace	string	Yes	Specify the path of a namespace using slashes (/) or colons (:). The number of namespaces in a module name is not limited.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies the module on which the command operates.

Return value

void

Example

```
// Create a namespace called Bob
addNamespace "Bob";

// Create a module name with multiple namespaces.
// Note Shogun doesn't separate the concept of module name vs. namespace.
addNamespace "Bob:Solving:Root:Spine:Head";
```

Additional information

Related commands

- | [getNamespace \(page 606\)](#)
- | [getNamespaces \(page 608\)](#)
- | [removeNamespace \(page 911\)](#)
- | [replaceNamespace \(page 927\)](#)

addParameter

Description

Summary

Creates a new dynamic parameter and optionally specifies a default value.

Details

Creates a new dynamic parameter and optionally defines the default value to be used for defining a template skeleton setup. Parameters are associated with a subject and can be used for markers and/or bones. Parameters can be shared across multiple objects.

Functional area

Parameters

Command syntax

Syntax

```
addParameter parameterName parameterValue(optional)[-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	Yes	Defines the name of the parameter to be created.
parameterValue	float	No	Defines the default value of the parameter

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Name of a character module in the scene to add parameters to.
solving				

Return value

void

Examples

```
//Create a parameter called HipLength and set its value to 6.0
addParameter "HipLength" 6.0;
```

Additional information

Related commands

- | [addStaticParameter \(page 125\)](#)
- | [getParameters \(page 633\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [selectByParameter \(page 968\)](#)
- | [setParameter \(page 1118\)](#)

addScript

Description

Summary

Add a script to Shogun's internal list of scripts.

Details

By default, Shogun fills its internal list with all of the hsl files found in the "script directories" you specify in your Preferences. However, if you wish to be able to call a script which isn't under one of those directories, then you must either use the [runScript \(page 941\)](#) command, or you must add the script to Shogun's internal list of scripts using addScript.

Note that if you wish to call the script from a hot key, then you should use this command rather than [runScript \(page 941\)](#).

Also note that if the file name of the script you add conflicts with an existing script or command in Shogun's internal list, then this command will fail. It will also fail if the script does not exist on the file system.

Functional area

System

Command syntax

Syntax

```
addScript "scriptPath"
```

Arguments

Name	Type	Required	Comments
scriptPath	string	yes	The full path of the script to add to Shogun's list of scripts. For paths, use forward-slashes instead of backslashes (e.g. C:/My Documents/MyScript.hsl)

Flags

None

Return value

void

Examples

```
// Iterate through all of the scripts in a directory and
// add them to Shogun's list of scripts
string $scripts[] = `getFileList "X:/My/Server/Scripts/" -pattern "*.hsl"`;
int $index, $count;
$count = `getCount $scripts`;
for( $index = 0; $index < $count; $index += 1 )
{
    addScript $scripts[ $index ];
}
```

Additional information

Related commands

- | [addScriptPath \(page 123\)](#)
- | [clearScriptPaths \(page 208\)](#)
- | [deleteScriptPath \(page 346\)](#)
- | [getScriptPaths \(page 670\)](#)
- | [runScript \(page 670\)](#)
- | [scriptExists \(page 952\)](#)
- | [setScriptPaths \(page 1151\)](#)

addScriptPath

Description

Summary

Add a directory to the list of "script directories" Shogun searches for scripts and pipelines.

Details

Add a directory to the list of "script directories" searched when Shogun builds its internal list of script and pipelines. By default, this command does not cause the internal list to get rebuilt. To force an immediate rebuild, specify the `-reparse` flag.

Functional area

System

Command syntax

Syntax

```
addScriptPath ScriptPath [-reparse]
```

Arguments

Name	Type	Required	Comments
path	string	yes	Path to add to set of search paths. File and directory paths must use forward slashes instead of back-slashes.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
reparse	0	—	—	If option is set the scripts and the pipelines from the directory will be reloaded

Return value

void

Examples

```
// Add a new script directory and re-load scripts and pipelines
addScriptPath "C:/My Documents/Shogun/MyScripts/" -reparse;
```

Additional information

Related commands

- [clearScriptPaths \(page 208\)](#)
- [deleteScriptPath \(page 346\)](#)
- [getScriptPath \(page 670\)](#)

addStaticParameter

Description

Summary

Adds a static parameter to a character.

Functional area

Parameters

Command syntax

Syntax

```
addStaticParameter parameterName parameterValue(optional) [-onMod string]
[-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	yes	Specifies the name of the parameter added
parameterValue	float	no	Specifies the value of the parameter added

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod		string	—	May be used to specify the character to which to add the parameter. Without this option, the selected character is used.
solving				Specifies if the parameter should be added to the solving setup instead of the labeling setup

Return value

void

addTab

Description

Summary

Add an additional tab to the tab controller.

Details

Adds an additional tab to the already created tab controller

Functional area

User Window

Command syntax

Syntax

```
addTab userControlID "itemString"
```

Arguments

Name	Type	Required	Comments
itemString	string	no	The name of the tab to be added
userControlId	int	yes	ID of user window to place the control on.

Flags

None

Return value

integer

Examples

```
// Create a Tab control and add two tabs
int $windowId;
int $tabControl;
int $firstTab;
int $secondTab;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

//create a tab control
$tabControl = `createTab $windowId`;

//add tabs
$firstTab = `addTab $tabControl "First"`;
$secondTab = `addTab $tabControl "Second"`;
```

Additional information

Related commands

- | [createTab \(page 300\)](#)
- | [deleteAllTabItems \(page 330\)](#)
- | [deleteTabItem \(page 352\)](#)
- | [findTabItem \(page 416\)](#)
- | [getTabItem \(page 697\)](#)
- | [getTabSelItem \(page 699\)](#)
- | [selectTabItem \(page 1007\)](#)
- | [setTabHandler \(page 1167\)](#)

addToClip

Description

Summary

Add specified or selected modules to specified or current clip.

Details

Use this command to add selected modules to the currently active clip or to add specific modules to specific clips. When modules are added to the specified/active clip they do not carry over any animation from any other clip. The modules exist with only their constant values.

Functional area

Data manipulators

Command syntax

Syntax

```
addToClip ["module" ...] [-toClip string]
```

Arguments

Name	Type	Required	Comments
module1	string	no	Name(s) of the module(s) to add to the clip. You may pass in a string array containing a list of modules to add to the clip. You can also pass in multiple strings. If no strings/arrays are specified, then all Modules will be added to the clip (or selected modules, if you specify the -selected flag.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
toClip	1	string	—	Name of the clip you want the modules to be added to. If not specified, then modules will be added to the active clip.

Return value

void

Examples

```
// create a Clip and a Marker
create Clip "clip1";
create Marker "marker1";

// create a 2nd Clip
create Clip "clip2";

// add marker1 to clip2
addToClip marker1 -toClip clip2;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)

-
- [removeLayer \(page 905\)](#)
 - [resetClip \(page 931\)](#)
 - [selectClipObjects \(page 979\)](#)
 - [setActiveClip \(page 1017\)](#)
 - [setActiveClip \(page 1017\)](#)
 - [setActiveLayer \(page 1020\)](#)
 - [tileClips \(page 1311\)](#)

alignAxis

Description

Summary

Align local axis orientation.

Details

The alignAxis commands provide maximum bone configuration flexibility when working with skeletons. Often it is desirable to be able to adjust bone axes individually in order to match the requirements of the environment into which the motion capture animations will be imported. Shogun can support bones with any orientation of the local axis; however, this command will always put one of the specified axes "down the bone" for twist purposes.

Functional area

Data manipulators

Command syntax

Syntax

```
alignAxis "x/-x/y/-y/z/-z" "x/-x/y/-y/z/-z" "x/-x/y/-y/z/-z" [-toSystem  
string string string] [-toNode string] [-toLine string string]
```

Arguments

A string composed of three letters, separated by spaces. Each component of the argument may be either X, Y, or Z.

The rotation axis indicated by the first letter in the argument, x axis if the argument is "x z z", for example, will be aligned with the vector from this joint to its first child joint.

The rotation axis indicated by the second letter in the argument, z axis if the argument is "x z z", will be aligned with the world axis indicated in the third letter of the argument (z).

The remaining axis is aligned according the right hand rule.

The arguments modify the joint orientation so that:

Name	Type	Required	Comments
x y z			The axis indicated here specifies the world axis that the second indicated joint axis will be aligned with
x y z			The axis indicated here is the axis of the joint that will be aligned with the world axis indicated in the third argument
x y z			The axis indicated here will be aligned with a vector drawn from the indicated joint to its child (i.e. down the twist axis of the bone.).

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
toSystem	3	string	toNode, toLine	Aligns the axis specified in the second letter of the argument to an axis of the coordinate system formed by the three given nodes, as specified by the third letter of the argument
toNode	1	string	toSystem, toLine	Aligns the axis specified in the second letter of the argument with an axis of the named node specified by the third letter of the argument
toLine	2	string	toSystem, toNode	—

Return value

void

Examples

```
alignAxis y z z;  
// Align a BoneNode's local axis as follows:  
// Y axis aligned with the vector from the indicated  
// node to its first child (i.e. down the bone).  
// Z axis aligned (as closely as is possible)  
// with the world Z axis.
```

Additional information

Related commands

- [snapToSystem \(page 1220\)](#)

amcExportOptions

Description

Summary

Set AMC export options for the next time an AMC file is exported.

Details

Functional area

File handling

Command syntax

Syntax

```
amcExportOptions [-removeGimbals boolean] [-writeAllDofs boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
removeGimbals	1	boolean	—	—
writeAllDofs	1	boolean	—	—

Return value

void

Examples

```
amcExportOptions -removeGimbals true;  
// remove gimbals next time an AMC is exported  
  
amcExportOptions -writeAllDOFs true;  
// write all channels not just the ones with DOFs on them
```

Additional information

Related commands

- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [mcpImportOptions \(page 798\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dImportOptions \(page 1375\)](#)
- | [xcplImportOptions \(page 1377\)](#)

animRange

Description

Summary

Set the play range.

Details

Set the play Range.

If `-start` is used then only the `inTime` should be specified, the end time of the play range remains unchanged from the existing.

If `-end` time is used then only `outTime` should be specified, the start time of the play range remains unchanged from the existing.

Otherwise both the start and end time must be specified.

Functional area

Playback control

Command syntax

Syntax

```
animRange inTime outTime [-r] [-start] [-end]
```

Arguments

Name	Type	Required	Comments
<code>inTime</code>	integer	No (dependant)	Starting Time
<code>outTime</code>	integer	No (dependant)	Ending Time

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
r	0	—	—	Start and end times specified are relative to the existing playrange start and end times
start	0	—	end	inTime is the start time, outTime is not specified
end	0	—	start	outTime is the end Time, inTime is not specified

Return value

void

Examples

```
// set the play Range to 50 -200
playRange 50 200;

// set the start of the play Range to 50
playRange 50 -start;

// set the end of the play Range to 200
playRange 200 -end;

// when existing play range start is frame 50
// sets the start to 150 ( 50 + 100 )
playRange 100 -r -start;
```

Additional information

Related commands

- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)

-
- | [playOptions \(page 834\)](#)
 - | [playRange \(page 836\)](#)
 - | [replay \(page 929\)](#)
 - | [setRangesFollowActiveClip \(page 1141\)](#)
 - | [setTime \(page 1173\)](#)
 - | [step \(page 1271\)](#)
 - | [stepKey \(page 1273\)](#)
 - | [stop \(page 1275\)](#)

appInfo

Description

Summary

Return Shogun application information.

Details

The appInfo profile command will return a string containing the Shogun application information (currently in use) that was queried depending, on the type of argument

Functional area

Interface

Command syntax

Syntax

```
appInfo "str" [-noFeedback]
```

Arguments

Name	Type	Required	Comments
>profile	str	Yes	Return the full path of the current profile
version	str	Yes	Return the version of the current executable including the SVN build number
exepath	str	Yes	Return the full path of the current executable
exeLocation	str	Yes	Return the folder of the current executable
computername	str	Yes	Return the computer name

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
noFeedback	0	—	—	Prevent command from printing to the command log.

Return value

string

Examples

```
string $t1 = `appInfo "profile"`;
string $t2 = `appInfo "version"`;
string $t3 = `appInfo "exepath"`;
string $t4 = `appInfo "exelocation"`;
string $t5 = `appInfo "user"`;
string $t6 = `appInfo "computername"`;
print $t1 $t2 $t3 $t4 $t5 $t6;

// Result may look like this
// Profile currently in use D:/Shogun/bin/Default.ini
// Version is: 1.2.158.34982
// Full path to exe: D:/Shogun/bin/Shogun.exe
// Location of exe: D:/Shogun/bin/
// Current user: user
// Computer name: VMS-DEV1
```

asfExportOptions

Description

Summary

Sets the file export options for the ASF file format.

Details

ASF is a skeleton file format. This format holds the skeleton definition for AMC file data. An ASF file can only contain one character hierarchy.

Functional area

File handling

Command syntax

Syntax

```
asfExportOptions [-writeAllDofs boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
writeAllDofs	1	Boolean	—	Ensures that any active bone node will contain all three degrees of freedom even if the bone was solved with just one or two DOFs active.

Return value

void

Examples

```
//set the ASF export to write all rotation DOFs.  
asfExportOptions -writeAllDofs true;
```

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [saveFile \(page 943\)](#)
- | [xcplImportOptions \(page 1377\)](#)
- | [x2dlImportOptions \(page 1375\)](#)

asin

Description

Summary

Returns the arc sine (aka inverse sine) of an input

Details

Arc sine is the inverse function for sine. That is, given a value for `val` in the domain $[-1.0, 1.0]$, it will return an angle `asin(val)` in degrees.

To illustrate, picture the unit circle (a circle of radius 1.0 centered about the origin). If you imagine a unit vector rotating inside of the unit circle, the y-coordinate of the endpoint of the vector is the `asin` function argument `val`. The angle that the vector makes with the positive x-axis is the function output `asin(val)`. As the vector rotates inside the circle, the y-coordinate of the vector must always be within the unit circle, so the input argument to `asin` (aka `val`) must be within $[-1.0, 1.0]$ to be meaningful.

Also, as $\sin(t)$ is doubly valued for t in $[-180, 180]$, the `asin` function is restricted to quadrant IV and quadrant I. This means that the unit vector rotating inside the unit circle is assumed to have a positive or 0 x-coordinate, therefore `asin(val)` always returns a value in $[-90, 90]$.

Functional area

Math

Command syntax

Syntax

```
asin value
```

Arguments

Name	Type	Required	Comments
value	float	yes	The domain of arc sine is $[-1.0, 1.0]$. Values of <code>val</code> outside of this domain are not meaningful to the function <code>asin</code> .

Flags

None

Return value

float

For an input argument `val` in `[-1.0, 1.0]`, returns a floating point angle (degrees) in the range `[-90.0, 90.0]`. For input arguments `t` in the ranges: `(1.0, Inf]`, `[-Inf, -1.0)` and `NaN`, `asin(t)` returns `NaN`.

Examples

```
float $t;
float $asin_of_t;
float $sin_of_asin_of_t;
// asin(1.0) is 90 degrees because sin(90 degrees) = 1.0
$t = 1.0;
$asin_of_t = asin($t);
$sin_of_asin_of_t = sin($asin_of_t);
print("t, asin(t), sin(asin(t)) = " + string($t) + ", " + string($asin_of_t) +
", " + string($sin_of_asin_of_t));
// 1.1 is not a meaningful input into asin and will return
// NaN (-1.#IND00)$t = -1.1;
$asin_of_t = asin($t);
$sin_of_asin_of_t = sin($asin_of_t);
print("t, asin(t), sin(asin(t)) = " + string($t) + ", " + string($asin_of_t) +
", " + string($sin_of_asin_of_t));
```

Additional information

Related commands

- [acos \(page 103\)](#)
- [atan \(page 148\)](#)
- [cos \(page 230\)](#)
- [sin \(page 1199\)](#)
- [tan \(page 1308\)](#)

assert

Description

Summary

Check for error condition.

Details

Provides a runtime test for error conditions. If the test returns true, the assert passes and does nothing. If the test returns false, the script terminates and an assertion failure is reported along with any number of additional arguments supplied to the assert statement after the test.

The additional arguments can be any string or value that can be printed, similar to the [print \(page 841\)](#) command.

Functional area

System

Command syntax

Syntax

```
assert booleanExpression [optionalMessage1] [optionalMessage2] ...
```

Arguments

Name	Type	Required	Comments
booleanExpression	boolean	yes	The condition to check for. If this condition evaluates to false, then the assert will fire and your script will fail.

Flags

None

Return value

void

Examples

```
int $i;  
// Do something intelligent to set the value of i  
$i = 400;  
  
// Check to make sure $i was set properly  
assert( $i == 400, "i was improperly set to ", $i );
```

Additional information

Related commands

■ [print \(page 841\)](#)

atan

Description

Summary

Returns the arc tangent (aka inverse tangent) of an input

Details

Arc tangent is the inverse function for tangent. Given a value for `val`, it will return an angle `atan(val)` in degrees.

To illustrate, picture the unit circle (a circle of radius 1.0 centered about the origin). Imagine a unit vector rotating inside of the unit circle. Draw a perpendicular line (tangent line) from the endpoint of the unit vector to closest point where the perpendicular line would intersect the x-axis. The length of that perpendicular (tangent) line is the input argument `val`. The angle in degrees between the positive x-axis and the unit vector is the output of the function `atan(val)`.

Functional area

Math

Command syntax

Syntax

```
atan value
```

Arguments

Name	Type	Required	Comments
value	float	yes	The domain of arc tangent is $[-\text{Inf}, \text{Inf}]$

Flags

None

Return value

float

Returns a floating point angle in degrees

Examples

```
float $temp = (atan (0.800));  
// declares a variable for the arc tan function  
  
print (string ($temp));  
// prints the output of the arc or inverse tan function
```

Additional information

Related commands

- | [acos \(page 103\)](#)
- | [asin \(page 144\)](#)
- | [cos \(page 230\)](#)
- | [sin \(page 1199\)](#)
- | [tan \(page 1308\)](#)

attach

Description

Summary

Attaches a module (typically a marker) to a bone.

Details

Creates a constraint between a module (typically a marker) and a bone. Also automatically adds parameters for the constraint offset to the constraint.

Used when setting up a skeleton to be solved or creating a VST.

The [createSkelScript \(page 291\)](#) command uses the attach command to create constraints when the `-solvers` flag is used.

Functional area

Skeletal solving

Command syntax

Syntax

```
attach "targetBone" "module1" "module2"... [-parent]
```

Arguments

Name	Type	Required	Comments
targetBone	String	yes	Name of the Bone the Modules should be attached to.
module1	String	yes	Name of a Module to attach to the bone.
marker2	String	no	Name of another Module to attach to the bone.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
parent	1	string	—	—

Return value

void

Examples

```
// Create a Bone and a Marker
create BoneNode "root";
create Marker "marker1";

// Attach the marker to the bone resulting in a new constraint.
attach root marker1;
```

Additional information

Related commands

■ [createSkelScript \(page 291\)](#)

autoCreateLabelingClusters

Description

Summary

Automatically creates labeling clusters.

Details

Searches a reconstructed ROM for groups of markers that are likely to be on a rigid labeling cluster.

Functional area

Labeling

Command syntax

Syntax

```
autoCreateLabelingClusters clusterFlexibility maxDistance minClusterSize
autolabelClusters
```

Arguments

Name	Type	Required	Comments
clusterFlexibility			Determines the likelihood of being a cluster
maxDistance			Determines the likelihood of being a cluster
minClusterSize			Determines the likelihood of being a cluster
autolabelClusters			Specify whether, after the cluster is created, the labeling clusters markers should be automatically labeled over the play range.

Flags

None

Return value

void

autoCreateSolver

Description

Summary

Creates and sets up a solver for the given character.

Details

To solve a skeleton, the bones of the skeleton must be added to a solver. Typically a solver is created for each character and all the bones of each character are added to the solver that represents that character. When the solve command is executed, a solver is automatically created for any characters in the scene with none of its bones already in a solver. This is done as a convenience to the user so they do not need to create and setup solvers themselves. This command allows the auto-creation of a solver to be performed explicitly.

Functional area

Skeletal solving

Command syntax

Syntax

```
autoCreateSolver characterName[-addSelectedBonesOnly]
```

Arguments

Name	Type	Required	Comments
characterName	String	Yes	Specifies the name of the character to create a solver for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
addSelectedBonesOnly	0	—	—	Specifies that only the selected bones should be added to the solver.

Return value

boolean

Examples

```
// Auto-create a solver for the Character named Bob.
autoCreateSolver "Bob";
```

Additional information

Related commands

- [solve \(page 1226\)](#)
- [solver \(page 1228\)](#)

autohideWindow

Description

Summary

Auto-hide a docking window (or pane)

Details

This command will auto-hide a docking window, or pane. Auto-hiding a pane will replace the window with a small button with some text and an icon on the edge of the main Shogun window.

Auto-hidden windows are useful in that they reduce visual clutter in your workspace. They allow quick access to the panes you want close by but not visible all the time. If you hover over the button, the pane will quickly show, allowing you to perform an operation. When you click away from the pane, the pane will disappear again.

Note that panes that are "floating" may not be auto-hidden; only docked panes may be auto-hidden.

Functional area

Interface

Command syntax

Syntax

```
autohideWindow "windowName"
```

Arguments

Name	Type	Required	Comments
windowName	string	true	The name of the window to auto-hide

Flags

None

Return value

void

Examples

```
// Auto-hide the "Attributes" window  
autohideWindow "Attributes";
```

Additional information

Related commands

- | [dockWindow \(page 361\)](#)
- | [floatWindow \(page 428\)](#)
- | [setWindowSize \(page 1191\)](#)
- | [showWindow \(page 1197\)](#)
- | [tabWindow \(page 1306\)](#)

autoLabel

Description

Summary

Autolabel trajectories

Details

Autolabeling is Shogun's default automated mechanism to assign unlabeled markers to the markers of a calibrated character. Autolabeling examines the positions of unlabeled markers in the frame and attempts to fit them to the marker sets of the characters loaded into the scene, based on the relative distances between marker pairs. The `autoLabel` command autolabels trajectories based on the frame range specified by the `rangeMode` option.

Labeling selected ranges requires specifying whether the labeling of markers will extend beyond the selected ranges, since trajectory lifetimes may extend beyond the specified start and end times of the selected ranges.

If extending the labeling to frames outside the selected ranges is desired, use the `selectedranges` mode; otherwise, use the `selectedrangesonly` mode, which will break the trajectories where they cross the start and end of the selected ranges.

<code>rangeMode</code> option	Description
<code>playrange</code>	Autolabel over the entire play range
<code>selectedranges</code>	Autolabel over selected ranges and extend labels to labeled trajectories beyond range of selection
<code>selectedrangesonly</code>	Autolabel over selected ranges only within the frames within the selected ranges, breaking trajectories
<code>currentframe</code>	Autolabels the current frame without breaking trajectories

Functional area

Labeling

Command syntax

Syntax

```
autoLabel [-rangeMode string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
rangeMode	1	string	—	<p>Set the range mode of the autolabel operation to one of the following values:</p> <p>playrange Autolabel the frames contained in the entire play range</p> <p>selectedranges Autolabel only the selected frames, extending labels to the entire trajectory including to frames outside the selected ranges</p> <p>selectedrangesonly Autolabel the selected frames, and do not extend application of labeling beyond the selected ranges, breaking trajectories if necessary</p> <p>currentframe Autolabel the trajectories in the current frame without breaking them</p>

Return value

void

Examples

```
// Autolabel the entire play range.
autoLabel -rangeMode "playRange";
```

Additional information

Related commands

- [autoLabelOptions \(page 161\)](#)

autoLabelOptions

Description

Summary

Sets the options on the autolabeler.

Details

The autolabel operation attempts to label all the markers of the subjects in the scene for all time, selected ranges, or the current frame. When doing so, the trajectories the autolabeler will consider can be specified based on selection or length(in frames).

The autoLabelOptions command allows these options to be set as well as additional options that account for the subject calibration being less then ideal.

Functional area

Labeling

Command syntax

Syntax

```
autoLabelOptions [-rangeMode string] [-minTrajLength integer] [-
jointPlacementSlack float] [-fixedSlack float] [-entranceThreshold float]
[-exitThreshold float] [-jointLmtSlack float] [-recalSlack float] [-
slackFactor float] [-selectedTraj boolean] [-solve boolean] [-
labelingClusterFlexibility float] [-restoreDefaultSettings]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
rangeMode	1	string	—	Specifies the time range the autolabeler will label.
minTrajLength	1	integer	—	Only trajectories that are this length or longer will get considered by the autolabeler.
jointPlacementSlack	1	float	—	Specifies the amount in mm the joint positions of the calibrated skeleton might be off by. Increase this value if you have low confidence in the joint positions of the skeleton.
fixedSlack	1	float	—	Specifies the amount in mm the markers positions (constraint offsets) of the calibrated skeleton might be off by. Increase this value if you have low confidence in the marker positions relative to the skeleton.
entranceThreshold	1	float	—	Specifies the percentage of markers that must get labeled for the autolabeler to consider the actor in the volume and switch from boot mode to tracking mode.

Name	Flag arguments	Argument type	Exclusive to	Comments
exitThreshold	1	float	—	Specifies the percentage of markers that get labeled for the autolabeler to consider the actor out of the volume.
jointLmtSlack	1	float	—	Specifies the amount in percentage to increase the joint range by. Increase this value if you think the ROM used to calibrate the skeleton did not have enough motion.
recalSlack	1	float	—	Specifies the amount in mm the marker covariance might be off by. Increase this value if you have low confidence in the maker covariances from the calibration.
slackFactor	1	float	—	This allows slight extra uncertainty in marker placement, as a proportion of the wobble observed in the subject calibration.
selectedTraj	1	boolean	—	Specifies that only selected trajectories should be considered by the autolabeler.
restoreDefaultSettings	0	—	—	Resets the autolabel options.
labelingClusterFlexibility		float		
solve		boolean		Solves the labeling skeleton after automatic labeling occurs.

Return value

void

Examples

```
// Set the autolabeler to only label the select time ranges and not  
// allow labels applied to extend beyond the selected ranges.  
autoLabelOptions -rangeMode "RangesOnly";
```

Additional information

Related commands

■ [autoLabel \(page 158\)](#)

autoVST

Description

Summary

Create a standard VST.

Functional area

Labeling

Command syntax

Syntax

```
autoVST
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

■ [autoVSTOptions \(page 166\)](#)

autoVSTOptions

Description

Sets options for autoVST command.

Functional area

Labeling

Command syntax

Syntax

```
autoVSTOptions [-actorName string] [-boneNamingScheme string] [-fingerMarkers string] [-middleWaistMarkersLocation string] [-globalPreScaleMethod string] [-manualScaleFactor float] [-labelOrder string] [-markerColorLeft vector] [-markerColorMiddle vector] [-markerColorRight vector] [-stickColorLeft vector] [-stickColorMiddle vector] [-stickColorRight vector] [-boneColor vector] [-autoBoneColor string] [-boneColorMult float] [-reset]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

■ [autoVST \(page 165\)](#)

axiomLabel

Description

Summary

Autolabel trajectories

Details

Alternative to [autoLabel \(page 158\)](#), axiomLabel performs autolabeling using the latest Axiom algorithms. This is the same labeling that is used by QuickPost, but enables the labeling to be run as a discrete step and with existing pre-labeled data.

Autolabels a capture over the entire play range unless the `-ranges` or `-currentFrame` flag is used. To set the options for axiom labeling, use the [axiomLabelOptions \(page 169\)](#) command.

Functional area

Data editing

Command syntax

Syntax

```
axiomLabel [-ranges] [-currentFrame]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	-	-	currentFrame	Autolabels only the selected range(s) of the current capture using Axiom algorithms.
currentFrame	-	-	ranges	Autolabels only the current frame of the current capture using Axiom algorithms.

Return value

void

Additional information

Related commands

■ [axiomLabelOptions \(page 169\)](#)

axiomLabelOptions

Description

Summary

Sets the offline Axiom labeling options.

Details

Functional area

Data editing

Command syntax

Syntax

```
axiomLabelOptions [-useLabelingClusters boolean] [-  
labelCompletenessEntranceTheshold float] [-useRobustBootting boolean] [-  
boottingQuality float] [-boottingVersusTracking float] [-trackingQuality  
float] [-smoothingFactor float] [-enforceJointRanges boolean] [-  
jointRangesSlack float] [-singlePass boolean] [-clearExistingLabels  
boolean] [-numThreads] [-reset] [-fromRTK] [-toRTK]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>useLabelingClusters</code>	1	boolean	—	Specifies whether labeling clusters are required to label subjects.
<code>labelCompletenessEntranceTheshold</code>	1	float	—	Defines the minimum ratio of labels to declare the subject entering the scene.
<code>useRobustBootng</code>	1	boolean	—	When set to True, clusters will have less impact on bootng, which may fix bootng problems related to incorrect cluster labeling, but at the expense of processing speed.
<code>bootngQuality</code>	1	float	—	Values lower than 0 allow lower quality bootng solutions. Values greater than 0 tighten the threshold.
<code>bootngVersusTracking</code>	1	float	—	Values lower than 0 favor rebooting. Values greater than 0 favor tracking.
<code>trackingQuality</code>	1	float	—	Values lower than 0 favor worse tracking solutions. Values greater than 0 tighten the exit thresholds.
<code>smoothingFactor</code>	1	float	—	Prior importance used by the kinematic fitter in tracking mode. Defines the stiffness of the subject.

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>enforceJointRanges</code>	1	boolean	—	If set to True, will try to find a labeling solution while enforcing the subject joint ranges.
<code>jointRangesSlack</code>	1	float	—	Values greater than 1.0 extend the calibrated joint ranges. Values lower than one tighten the joint ranges.
<code>singlePass</code>	1	boolean	—	When set to true, labeling is performed in a single pass only.
<code>clearExistingLabels</code>		boolean		
<code>numThreads</code>	1	float	—	
<code>reset</code>				
<code>fromRTK</code>				
<code>toRTK</code>				

Return value

void

Additional information

Related commands

■ [axiomLabel \(page 167\)](#)

bakeData

Description

Summary

Copies all data from the currently active clip into the target clip, for the selected objects.

Details

bakeData is generally used to overwrite unwanted data modifications, copying the active clip data to the target clip (if the target clip is not specified then it bakes down to the next clip).

The command enables you to do restore to the character a previously saved state of animation data long after your levels of undo have run out.

Functional area

Data editing

Command syntax

Syntax

```
bakeData ["targetClip"] [-all] [-ranges] [-percent float] [-mustHaveKeys]
[-selectedChannels]
```

Arguments

Name	Type	Required	Comments
targetClip	string	no	Name of target clip.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0		ranges	Performs the copy on all objects in the scene.
ranges	0		all	Only copies data over the currently selected time ranges.
percent	1	float	—	Takes a float, between 0 and 100, indicating the percent difference to bake. 100 bakes normally.
mustHaveKeys	0		—	Module must have a key on both clips for data to be baked at any given frame.
selectedChannels				

Return value

void

Examples

```
// Load suitable Shogun file containing at least two subjects before running the
following
selectRange -all;
filter 0.15 35;
selectRange 380 450;
bakeData -ranges;

// In this example, a filter is applied to all
// of the markers pertaining to the actor.
// With all markers still selected, a range ( 380 - 450 )
// is selected, and the second figure data is baked
// to the active figure over the selected range.
// Make sure your second figure is turned on
// in the Perspective view (ALT + RMB, Show second figure)
// and notice that at frame 380 the first and second figure data
// become one and the same; original data having been baked
```

```
// back onto the first figure. At frame 451, the first  
// figure data snaps back to its filtered state.
```

Additional information

Related commands

- [copyData \(page 222\)](#)

bakeLengths

Description

Summary

Takes average position value of a node over all time, clears all the keys, and sets the constant value to that average.

Details

Takes average position value of a node over all time, clears all the keys, and sets the constant value to that average. Useful in finding out the average position of a node

Functional area

Data editing

Command syntax

Syntax

```
bakeLengths
```

Arguments

None

Flags

None

Return value

void

Examples

```
//bakeLengths on all markers  
selectByType Marker;  
bakeLengths;
```

Additional information

Related commands

- [setBoneLength \(page 1029\)](#)

breakTangents

Description

Summary

Break the tangent on a selected key into two separate parts or handle manipulators.

Details

Break the tangent on a selected key into two separate parts or handle manipulators. These can then be used to adjust curve slopes independently on either side of the key. This allows steeper gradients to be achieved than is possible with a single tangent, where both sides move together.

Functional area

Data manipulators

Command syntax

Syntax

```
breakTangents
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

- [createTangents \(page 303\)](#)
- [unbreakTangents \(page 1321\)](#)

bvhExportOptions

Description

Summary

Sets the file export options for the BVH file format.

Details

BVH is a skeleton animation file format. The format holds the skeleton definition and animation data for a given trail. A BVH file can only contain one character hierarchy and will not allow local transforms in the base pose.

Functional area

File handling

Command syntax

Syntax

```
bvhExportOptions [-writeDofs boolean] [-padStartFrames boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
writeDofs	1	boolean	—	Ensures that any active bone node will contain all three degrees of freedom even if the bone was solved with just one or two DOFs active.

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>padStartFrames</code>	1	boolean	—	Creates zero value frames of data until the first frame of data specified in Shogun. For instance, if the Shogun scene starts at frame 60, using this option will insert 59 blank frames into the BVH file. This option is useful if you are trying to ensure temporal (or time) relationship between a BVH file and other files or environments that maintain explicit frame ranges.

Return value

void

Examples

```
//set the BVH export to write all rotation DOFs and pad with
// zeros up to the start frame.
bvhExportOptions -writeDofs ture -padStartFrames ture;
```

Additional information

Related commands

- [amcExportOptions \(page 135\)](#)
- [c3dExportOptions \(page 181\)](#)
- [cpExportOptions \(page 232\)](#)
- [fbxExportOptions \(page 376\)](#)
- [fbxImportOptions \(page 380\)](#)
- [loadFile \(page 779\)](#)
- [mcpExportOptions \(page 796\)](#)
- [mcplImportOptions \(page 798\)](#)
- [saveFile \(page 943\)](#)
- [x2dlImportOptions \(page 1375\)](#)
- [xcplImportOptions \(page 1377\)](#)

c3dExportOptions

Description

Summary

Set C3D Export options

Details

Use this command to set options for Shogun's C3D exporter. Typically you won't need to modify the C3D exporter settings as Shogun's default settings should work in most situations. Note that these options are persistent, meaning they persist until you change them again.

Functional area

File handling

Command syntax

Syntax

```
c3dExportOptions [-preserveGaps boolean] [-collapseSubjects boolean] [-real boolean] [-writeTimeCode boolean] [-writeUnlabeled boolean] [-removeLeadingUnderscore boolean] [-decFormat boolean] [-filterMin boolean] [-filterMax boolean] [-minKeys integer] [-maxKeys integer] [-overrideHeaderRate boolean] [-headerRate float]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
preserveGaps	1	boolean	—	Specifies that the exporter maintains data gaps on markers in c3d files. When false, the exporter writes keys out for gaps using interpolated values from the Markers' Translation channels.
collapseSubjects	1	boolean	—	Specifies that the exporter writes out Marker names as CharacterName::MarkerName. When false, the exporter will only write out MarkerName.
real	1	boolean	—	Specifies that the exporter writes out Translation values as 4 byte floating point values. Otherwise, the exporter will write out 2 byte integers, and one 4 byte scale value which each value should be scaled by upon import. (Setting this flag to false may affect data integrity).
writeTimeCode	1	boolean	—	Specifies that the exporter writes out the SMPTE_Offset attribute value for the active clip into the c3d files TIMECODE section. When false, no TIMECODE section will be written.
writeUnlabeled	1	boolean	—	Specifies that the exporter writes out unlabeled Trajectories. When false, unlabeled Trajectories will be not be written out.

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>removeLeadingUnder score</code>	1	boolean	—	Specifies that the exporter remove underscore (<code>_</code>) characters from Character names upon export. This may be necessary if you imported a C3D that had Characters whose names began with a digit (causing Shogun to prepend an underscore to the Character names) and you want to preserve the original Character name upon export.
<code>decFormat</code>	1	boolean	—	Specifies the byte ordering for floating point values written to the C3D. You should always leave this on.
<code>filterMin</code>	1	boolean	—	Specifies that the exporter skip writing of Markers with less than N Translation keys, where N is the value specified by the <code>-minKeys</code> flag.
<code>filterMax</code>	1	boolean	—	Specifies that the exporter skip writing of Markers with more than N Translation keys, where N is the value specified by the <code>-maxKeys</code> flag.
<code>minKeys</code>	1	integer	—	Used in conjunction with the <code>-filterMin</code> flag. See <code>-filterMin</code> above.
<code>maxKeys</code>	1	integer	—	Used in conjunction with the <code>-filterMax</code> flag. See <code>-filterMax</code> above.
<code>overrideHeaderRate</code>	1	boolean	—	Specifies that the exporter should write out a frame rate in the C3D header as specified by the <code>-</code>

Name	Flag arguments	Argument type	Exclusive to	Comments
				headerRate flag. When false, the scene rate will be used. Note that no resampling to the data happens. The exporter simply indicates that the data in the file is at a different rate than it really is. Use with care.
headerRate	1	float	—	Used in conjunction with the -overrideHeaderRate flag. See -overrideHeaderRate above.

Return value

void

Examples

```
// This will turn on timecode exporting
c3dExportOptions -writeTimeCode true;
```

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [mcpImportOptions \(page 798\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dImportOptions \(page 1375\)](#)
- | [xcplImportOptions \(page 1377\)](#)

calcIntersection

Description

Summary

Calculates the intersection point of three or more spheres.

Details

Calculates the intersection points of three or more spheres using trilateration.

Takes as input a vector array (the world space locations of the sphere centers) and a float array (the radii of the spheres) and returns a vector array representing the points of intersection of the spheres. This may be one or more points depending on the number and arrangement of the spheres.

Can be useful in gap filling. Given a marker whose trajectory is gapped, it is possible to select a number of pairs of markers each of which consists of the gapped marker and another marker whose distance is consistent from the first throughout the desired range.

Each pair will then define a sphere whose center is the position of the second (ungapped) marker and with radius equal to its distance from the gapped marker.

The marker positions and distances can be used as input for the calcIntersection command which will return the intersections points of the spheres defined by the selected marker pairs. Each returned intersection point is a candidate for the gapped marker position at a given frame. Performing this process for every frame of a gap would allow the gap to be filled.

Results can vary widely. For good results, high quality, stable input data is required. The results of this command should be used with caution.

Functional area

Math

Command syntax

Syntax

```
calcIntersection "position array" "radius array"
```

Arguments

Name	Type	Required	Comments
radius	array	yes	Radii of the spheres
position	array	yes	World space locations of the sphere centers

Flags

None

Return value

vector array

Additional information

Related commands

- | [copyPattern \(page 226\)](#)
- | [fillGaps \(page 389\)](#)
- | [getDistance \(page 507\)](#)

calibrateCharacter

Description

Summary

Calibrates characters.

Details

If no characters are selected, calibrates all characters in the scene. If one or more characters are selected, calibrates each one. Calibration uses the entire play range unless the `-ranges` flag is used. See [calibrateCharacterOptions \(page 189\)](#) command to set calibration options.

Functional area

Labeling

Command syntax

Syntax

```
calibrateCharacter [-ranges]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	0	—	—	Performs the calibration only over the selected time ranges.

Return value

void

Examples

```
// Calibrate all characters in the scene.  
calibrateCharacter;
```

Additional information

Related commands

■ [calibrateCharacterOptions \(page 189\)](#)

calibrateCharacterOptions

Description

Summary

Sets the character calibration options.

Details

The character calibrator can adjust parameter values (affecting constraints and bones using parameters), preferred pose, joint range, and marker covariance.

The `-calibrationMode` flag allows the number of those properties that will be altered to be set. The importance of the rest importance, constraint offsets, and bone parameters, can also be specified. For instance, if you know the bone lengths of the skeleton to be calibrated are already very accurate (either due to manual setup or a previous calibration) you may decide to either not calibrate bone length by adding the `-calibrationMode "markersOnly"` option or you may put a high importance on the bone lengths so they are not greatly changed by setting the segment importance higher.

Functional area

Labeling

Command syntax

Syntax

```
calibrateCharacterOptions [-restImportance float] [-markerImportance float] [-segmentImportance float] [-quality string] [-activeFrames integer] [-calibrationMode string] [-statsMode string] [-autoSaveVsk boolean] [-autoSavePreCalVDF boolean] [-overwriteRangeTemplates boolean] [-showCovariance boolean] [-restoreDefaultSettings]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
restImportance	1	float	—	Specifies the strength of the restImportance during the calibration.
markerImportance	1	float	—	Specifies the importance of the constraint parameters during the calibration.
segmentImportance	1	float	—	Specifies the importance of the bone parameters during the calibration.
quality	1	string	—	Speed vs. quality tradeoff. Options are: fast, normal, accurate.
activeFrames	1	integer	—	Specifies how many frames the calibrator samples. The default is 150. The minimum is 100. This number of frames is only for part of the calibration. Other parts of the calibration sample more frames and cannot be adjusted.
calibrationMode	1	string	—	Specifies what the calibrator will change; everything (full), markersOnly, or preferred pose, joint range, and marker covariance (statsOnly).
statsMode	1	string	—	Has options for setting how statistics are used.
autoSaveVsk	1	boolean	—	When set to true a vsk file is automatically saved to disk after calibration is complete.
autoSavePreCalVDF		boolean		Saves a VDF of the scene before calibration.

Name	Flag arguments	Argument type	Exclusive to	Comments
overwriteRangeTemplates		boolean		
showCovariance	1	boolean		When set to true, labeling constraint covariance is automatically displayed after calibration.
restoreDefaultSettings	0	—	—	Resets the options to their default settings.

Return value

void

Examples

```
// Given a skeleton setup with already good bone lengths set the
// calibrator options so the bones will not be altered during
// calibration.
calibrateCharacterOptions -calibrationMode "markersOnly";

// Now calibrate the selection
calibrateCharacter;
```

Additional information

Related commands

- [calibrateCharacter \(page 187\)](#)
- [calibratePropOptions \(page 195\)](#)

calibrateLabelingCluster

Description

Summary

Calibrates the selected labeling cluster.

Functional area

Labeling

Command syntax

Syntax

```
calibrateLabelingCluster
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

■ [calibrateLabelingClusterOptions \(page 193\)](#)

calibrateLabelingClusterOptions

Description

Summary

Set options for calibrateLabelingCluster command.

Functional area

Labeling

Command syntax

Syntax

```
calibrateLabelingClusterOptions [-minMarkerStdDev float] [-  
motionCoherenceThreshold float] [-restoreDefaultSettings]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

■ [calibrateLabelingCluster \(page 192\)](#)

calibrateProp

Description

Summary

Calibrates the selected prop(s).

Details

The character node of the prop(s) should be selected before executing.

By default, the entire play range is used, but the ranges option may be used to calibrate using only a range of time.

Prop calibration is not necessary, but may be useful for props that have a bit of flex or whose markers are less rigid than ideal.

Functional area

Labeling

Command syntax

Syntax

```
calibrateProp [-ranges]
```

Arguments

None

Flags

None

Return value

void

calibratePropOptions

Description

Sets prop calibration options.

Functional area

Labeling

Command syntax

Syntax

```
calibratePropOptions [-restImportance float] [-markerImportance float] [-segmentImportance float] [-quality string] [-activeFrames integer] [-calibrationMode string] [-statsMode string] [-autoSaveVsk boolean] [-autoSaveVss boolean] [-autoSavePreCalHDF boolean] [-overwriteRangeTemplates boolean] [-showCovariance boolean] [-restoreDefaultSettings]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

■ [calibrateCharacterOptions \(page 189\)](#)

cameraView

Description

Summary

Set 3D workspace view pane properties.

Details

The cameraView command provides navigational flexibility within the 3D working environment. You can also decide whether to show certain elements of the scene like the active clip only, floor grid lines, etc.

Here are the meanings of the `-track` flag values:

Value	Result
<code>all</code>	Camera tracks all nodes in the scene
<code>selected</code>	Camera tracks selected nodes in the scene and changes the objects it tracks as you change selection
<code>primary</code>	Camera tracks the primary selected node in the scene and changes the object it tracks as you change the primary selection
<code>objects</code>	Camera tracks the selected nodes in the scene and continues to track them until you call <code>cameraView -track "objects"</code> again

Functional area

Interface

Command syntax

Syntax

```
cameraView [-track string] [-showAllClips boolean] [-floor boolean] [-axis boolean] [-zUp boolean] [-pointSize float] [-snapSelected] [-snapPrimary] [-connectLines boolean] [-resetLines] [-resetCamera] [-switchCamera string] [-levelCamera] [-trails boolean] [-pastTrails integer] [-futureTrails integer] [-allViews] [-inactiveErrorLines boolean] [-gapDim integer] [-keyMode boolean] [-gapConnections boolean] [-useClipColor boolean] [-halo boolean] [-gridSize float] [-gridSizeX float] [-gridSizeZ float] [-gridSpacing float] [-displayText boolean] [-displayCharacterText boolean] [-gapSelect boolean] [-lock boolean] [-contributionMode boolean] [-markerRadius boolean] [-displayTrajCount boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
track	1	string	—	Set the object tracking type. Valid types are "none", "all", "selected", "primary", "objects", and "cycle". See Details section for meaning of each type.
showAllClips	1	boolean	—	Show all clips or just the active clip in the current view. Objects of inactive clips will display in their clip color rather than the objects' own color.
floor	1	boolean	—	Turn floor grid display on/off
axis	1	boolean	—	Turns the world axis indicator in the bottom left of the 3D View on and off

Name	Flag arguments	Argument type	Exclusive to	Comments
zUp	1	boolean	—	Set Shogun Post to operate in Z-Up mode (Y-Up if turned off).
pointSize	1	float	—	Set marker display size in pixels.
snapSelected	0	—	—	Snap view to center of selected items.
snapPrimary	0	—	—	Snap view to center of primary selected item.
connectLines	1	boolean	—	Turn display of marker connectivity on/off.
resetLines	0	—	—	Clears marker connecting lines on selected markers.
resetCamera	0	—	—	Resets the cameras view to it's default state.
switchCamera	1	string	—	To switch from one camera to another, default switch to the "view" switch to the default view camera.
levelCamera	0	—	—	Removes any roll from the camera.
trails	1	boolean	—	Displays a 3D representation of marker trajectories. Turns -pastTrails and -futureTrails on and off.
pastTrails	1	integer	—	Number of frames to display for past trails.
futureTrails	1	integer	—	Number of frames to display for future trails.

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>allViews</code>	0	—	—	Specifies that other flags operate on all 3D views rather than just the active 3D view
<code>inactiveErrorLines</code>	1	boolean	—	By default, error lines will not be drawn for inactive clips, unless this flag is enabled.
<code>gapDim</code>	1	integer	—	Takes an integer argument between 0 and 100, specifying percentage a marker should be dimmed (made transparent) if it is on a gap.
<code>keyMode</code>	1	boolean	—	Mode where the keys on trails are selectable and editable from within the 3D View.
<code>gapConnections</code>	1	boolean	—	Controls whether marker connecting lines are drawn between markers that are on a gap.
<code>useClipColor</code>	1	boolean	—	Specifies that objects in the active clip draw in the color of the active clip rather than their own object color.
<code>halo</code>	1	boolean	—	Specifies that a "halo" be drawn around the currently selected "label" in the Labeling window.
<code>gridSize</code>	1	float	—	Specifies the number of meters squared that are displayed in the 3D View, for example, if <code>gridSize</code> = 10, the resulting grid would be 10m x 10m.

Name	Flag arguments	Argument type	Exclusive to	Comments
gridSizeX	1	float	—	Specifies the X extent from the origin (in meters) of the grid that is displayed in the 3D view.
gridSizeZ	1	float	—	Specifies the Y or Z extent from the origin (in meters) of the grid that is displayed in the 3D view.
gridSpacing	1	float	—	Specifies the spacing (in meters) between the lines of the grid that is displayed in the 3D view.
displayText	1	boolean	—	Specifies whether the "Heads Up Display" text be rendered onto the 3D view.
gapSelect	1	boolean	—	Specifies whether Markers can be selected from within the in the 3D view if they are on a gap.
lock	1	boolean	—	Turns off the 3D View. Useful if you want to prevent the 3D view from drawing to boost real time processing power.
contributionMode	1	boolean	—	Specifies that Markers /Trajectories should draw rays to the selected OpticalCameras that were used in creating them.

Return value

void

Examples

```
// This command puts Shogun Post's active 3D view in a mode in which the
// currently selected objects are always centered in the view.
cameraView -track selected;

// Will toggle Shogun Post between Z-up versus Y-up
cameraView zUp toggle;
```

Additional information

Related commands

- | [camerasView \(page 202\)](#)
- | [graphView \(page 730\)](#)
- | [viewLayout \(page 1342\)](#)

camerasView

Description

Summary

Set Cameras view pane properties

Details

The camerasView commands allow showing certain elements of the 2D cameras view like blobs, centroids, threshold map, set cameras' settings, etc.

Functional area

Interface

Command syntax

Syntax

```
camerasView [-alignWorldVertical boolean] [-zoomToFit] [-tileMode
boolean] [-primarySelectFull] [-allViews] [-showCircles boolean] [-
showBlobs boolean] [-showThresholdMaps boolean] [-showThresholdGrid
boolean] [-paintMode boolean] [-setPaintValue integer] [-showTypes
boolean] [-paintTypeRectangle boolean] [-showSettings boolean] [-
showNames boolean] [-showMarkers boolean] [-connectLines boolean] [-
showAllClips boolean] [-show2DTracks boolean] [-set2DTrackLength integer]
[-showCentroidCount boolean] [-showFitMethod boolean] [-
showSelectedCircles boolean] [-show3D boolean] [-distort3D boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
alignWorldVertical	1	boolean	—	Set on/off/toggle rotation mode for the selected cameras. If the mode is turned on then the camera view can be rotated using left mouse button. If not, the camera view cannot be moved.
zoomToFit	0	—	—	Zoom to fit the view of the selected cameras
tileMode	1	boolean	—	Show all cameras view or only view for selected camera
primarySelectFull	0	—	—	Reserved
allViews	0	—	—	If not specified all options are applied only to the active view. Otherwise options are applied on all Cameras views
showCircles	1	boolean	—	Show/Hide the centroids on the camera view
showBlobs	1	boolean	—	Show/Hide the blobs on the camera view
showThresholdMaps	1	boolean	—	Show/Hide the threshold map for the cameras
showThresholdGrid		boolean		
paintMode	1	boolean	—	Set the paint mode for the selected cameras. In this mode the user can

Name	Flag arguments	Argument type	Exclusive to	Comments
				manually edit the threshold map. If the mode is turned on, the threshold map is also on.
setPaintValue	1	integer	—	Set current value for threshold painting. The value's range is 0-255. Setting the value to 0 results in no paint.
showTypes	1	boolean	—	Show/Hide the type of the cameras
paintTypeRectangle	1	boolean	—	Set rectangle mode when painting the threshold map. In this mode user paints the map by selecting and dragging the mouse between two points. The threshold for selected rectangle will be set to current value
showSettings	1	boolean	—	Show/Hide settings for the selected cameras. User can change the values for "Strobe Intensity", "Threshold", "Gain", "Circularity"
showNames	1	boolean	—	Show/Hide the names of the cameras
showMarkers	1	boolean	—	—
connectLines	1	boolean	—	—
showAllClips	1	boolean	—	—
show2DTracks	1	boolean	—	—
set2DTrackLength	1	integer	—	—

Name	Flag arguments	Argument type	Exclusive to	Comments
showCentroidCount		boolean		
showFitMethod		boolean		
showSelectedCircles		boolean		
show3D		boolean		
distort3D		boolean		

Return value

void

Examples

```
// In this example the camera is selected, the camera's settings are
// changed and threshold map is prepared for drawing
setPrimary MX40_1234;
camerasView -tileMode off;
camerasView -showSettings toggle;setProperty "Strobe_Intensity" 0.739458 -onMod
MX40_252;
setProperty "Threshold" 0.350904 -onMod MX40_252;
setProperty "Gain" 3 -onMod MX40_252;
setProperty "Circularity" 0.359940 -onMod MX40_252;
camerasView -showThresholdMaps toggle;
camerasView -paintMode toggle;
camerasView -paintTypeRectangle toggle;
camerasView -setPaintValue 255;
// draw the threshold map
```

Additional information

Related commands

- | [cameraView \(page 196\)](#)
- | [graphView \(page 730\)](#)
- | [viewLayout \(page 1342\)](#)

clearLog

Description

Summary

Clears the Log window

Details

Clears the command Log window.

Functional area

System

Command syntax

Syntax

```
clearLog
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Clear the command log window so the print statements we
// execute below will be the only things showing in the
// command log window.
clearLog;
print "Hello world";
print "Good-bye world";
```

Additional information

Related commands

■ [listCommands \(page 773\)](#)

clearScriptPaths

Description

Summary

Clear all items from the list of directories searched by Shogun Post to generate the internal list of scripts and pipelines.

Details

Clear all items from the list of directories searched by Shogun Post to generate the internal list of scripts and pipelines. This list can also be viewed at the top of the **Directories** tab in the **Preferences** dialog.



Caution

Removing the default paths may cause some UI buttons to fail if the internal list of scripts and pipelines is rebuilt by reparsing the script directories.

Functional area

System

Command syntax

Syntax

```
clearScriptPaths;
```

Arguments

None

Flags

None

Return value

Void

Examples

```
// Clear existing script directories and replace with a new script
// directory and re-load scripts and pipelines
clearScriptPaths;
addScriptPath "C:/My Documents/Shogun/MyScripts/" -reparse;
```

Additional information

Related commands

- | [addScriptPath \(page 123\)](#)
- | [deleteScriptPath \(page 346\)](#)
- | [getScriptPaths \(page 670\)](#)

client

Description

Summary

Turns the client on and off

Details

This command allows you send remote script commands.

Functional area

Remote control

Command syntax

Syntax

```
client on/off/toggle [-port string] [-ip string]
```

Arguments

Name	Type	Required	Comments
on/off/toggle	boolean	yes	Turn the client on or off

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
port	1	string	—	Specifies the port to use for the network socket connection
ip		string		Specifies the IP address

Return value

void

Examples

```
// This command will enable the sending of commands to a remote server
client on -port 4344;
```

Additional information

Related commands

- [remoteControl \(page 897\)](#)
- [sendRemote \(page 1013\)](#)
- [server \(page 1015\)](#)

closeScript

Description

Summary

Close a script.

Details

Closes the script in which it is used or the script currently loaded into the script editor. A Save prompt is displayed if changes to the script have not been saved.

Functional area

Interface

Command syntax

Syntax

```
closeScript
```

Arguments

None

Flags

None

Return value

void

Examples

```
//selects all the bones in the current clip and closes the script  
selectByType Bone;  
closeScript;
```

Additional information

Related commands

- [saveScript \(page 945\)](#)

collapse

Description

Summary

Collapse the data of one duplicate hierarchy onto the other.

Details

Given two selected modules, it will copy keys for the source module and all its descendants to the target module (the primary selection) for the active clip. There must be only 2 selected modules in the scene, and an active clip. Useful when only one hierarchy is needed to represent the data within the file.

Functional area

Data editing

Command syntax

Syntax

```
collapse
```

Arguments

None

Flags

None

Return value

void

Examples

```
//import an ASF/AMC pair twice
//make sure and set the import type to createNewAlways

//select Characters
select Actor_2 Actor_1;
collapse;
```

Additional information

Related commands

■ [copyClip \(page 219\)](#)

compareModules

Description

Summary

Compare two same-type module attributes.

Details

Allows attribute comparison of two modules of the same type (bones, constraints, markers). Differences are reported in the log with minimal detail.

Functional area

Testing

Command syntax

Syntax

```
compareModules "module1" "module2" [-ignoreModuleAttributes];
```

Arguments

Name	Type	Required	Comments
modules	string	true	The names of the modules to compare.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ignoreModuleAttributes	0	—	—	<p>An attribute in Shogun Post that points to a module in the scene. The ignoreModuleAttributes flag can be used to skip comparing attributes where comparison is likely to fail due to the attributes pointed to by two modules not being the exact same module, even if their name differs. For example, for QA purposes, you might want to write a script to compare the hierarchy of an actor to a copy of itself.</p> <p>Character names must differ, so you could just skip checking the character node, but just about all modules parented to each character and their names will be the same between the two copies. So, you could write a script that iterates over all the pairs of modules and calls compareModules.</p> <p>If you're checking, say, a constraint, the source attribute will point to a marker. While that will be a marker of the same name in both copies, it won't be exactly the same marker, so the comparison will fail unless - ignoreModuleAttributes is used.</p>

Return value

string

Examples

```
compareModules "RightUpLeg" "LeftUpLeg";
```

When the above is entered into the Script Editor and the script is run, the Log displays the following:

```
ERROR: Attribute Name on RightUpLeg does not have the same data as attribute  
Name on LeftUpLeg.
```

```
ERROR: Attribute Pre_Translation on RightUpLeg does not have the same data as  
attribute Pre_Translation on LeftUpLeg.
```

```
ERROR: Attribute Joint_Range on RightUpLeg does not have the same data as  
attribute Joint_Range on LeftUpLeg.
```

```
ERROR: Attribute Joint_Covariance_Matrix on RightUpLeg does not have the same  
data as attribute Joint_Covariance_Matrix on LeftUpLeg.
```

```
ERROR: Attribute Parameter_PositionX on RightUpLeg does not have the same data  
as attribute Parameter_PositionX on LeftUpLeg.
```

```
ERROR: Attribute Preferred_Pose on RightUpLeg does not have the same data as  
attribute Preferred_Pose on LeftUpLeg.
```

copyClip

Description

Summary

Copies specified or selected modules with the animation of the current clip to a specified clip.

Details

The copyClip command can be used to copy either selected modules and data to a specified clip or copy specific modules and data to a specified clip. copyClip only copies modules and data from the currently active clip.

Functional area

Data manipulators

Command syntax

Syntax

```
copyClip "clipName" ["module" ...][-selected] [-create] [-leaveTimeAlone]
```

Arguments

Name	Type	Required	Comments
clipName	string	yes	Name of the clip you will be copying module and data to. If the <code>-create</code> flag is specified, then a clip with this name will be created.
module1	string or string array	no	Name of the module whose animation data to copy. You may pass in a string array containing a list of modules. You can also pass in multiple strings. If no strings/arrays are specified, then all modules will be copied (or selected modules, if you specify the <code>-selected</code> flag.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selected	0	—	—	If specified, and no arguments are passed in to the command, the selected modules will have their animation data copied
create	0	—	leaveTimeAlone	If specified, a new clip will be created as the target clip, with the name provided by the first argument to the command
leaveTimeAlone	0	—	create	If specified, the target clip will not have it's time related attributes changed (Start_Frame, Clip_Offset, and Duration)

Return value

void

Examples

```
// create a Clip and a Marker
create Clip "clip1";
create Marker "marker1";

// animate the marker
// to represent data being copied
setTime 0;
setKey Translation 0 0 0;
setTime 20;
setKey Translation 100 100 100;

// create a 2nd Clip
// copy marker1 to clip2
copyClip clip2 marker1 -create;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

copyData

Description

Summary

Copy data from one object's named output to a named input on one or more different objects.

Details

copyData is a one-time operation that copies the transformation data in the outputs of a source object and writes it to the inputs of one more target object for the entire animation range. When using copyData, the target node will have curves that may be edited.

Functional area

Data editing

Command syntax

Syntax

```
copyData module1.outputName module2.inputName1 ...
```

Arguments

Name	Type	Required	Comments
module1.outputName	string	yes	Target node(s) and output channel
module2.inputName1	string	yes	Source node and input channel
More target.channel pairs	string	no	Additional targets for the copy operation can be specified

Flags

None

Return value

void

Examples

```
// This command copies the translation from a marker called "STRN" to
// another scene object called "Prop1".
copyData "STRN.Translation" "Prop1.Translation";

// This command copies the translation from a marker called "STRN" to
// 2 other scene objects called "Prop2" and "Prop3".
copyData "STRN.Translation" "Prop2.Translation" "Prop3.Translation";
```

Additional information

Related commands

- | [bakeData \(page 172\)](#)
- | [copyKeys \(page 224\)](#)
- | [copyPattern \(page 226\)](#)
- | [parent \(page 826\)](#)
- | [pasteKeys \(page 828\)](#)
- | [snapTo \(page 1207\)](#)

copyKeys

Description

Summary

Copies all the selected keys to the clipboard.

Details

copyKeys may only be used under these conditions: a) exactly one module is selected, and b) a one channel selection is made within only one channel of the module.

For example, if BoneNode "right_foot" is selected, you may copy keys from Translation:X or Translation:Y or Translation:Z; but not from Translation:X and Translation:Y and/or Translation:Z.

Functional area

Data editing

Command syntax

Syntax

```
copyKeys [-all] [-ranges]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	ranges	Copies all the data for the controller to the clipboard
ranges	0	—	all	Copies the selected ranges to the clipboard

Return value

integer

Returns true if successful, false if the operation could not be performed.

Examples

```
// Select the Y Rotation channel of the module right_foot,
// and copy all the keys within the selected range to the clipboard.
select right_foot;
selectProperty "RotationY";
copyKeys -ranges;
```

Additional information

Related commands

■ [pasteKeys \(page 828\)](#)

copyPattern

Description

Summary

Copy transform pattern (not explicit values) from the selected nodes to primary selected node within the selected time ranges.

Details

When applying copyPattern, information is copied to the primary selection from the other members of the current selection. This command is not exactly a copy keys function, nor is it purely a curve interpolation function.

The command will respect the in and out points defined by the range of keys selected, and will smoothly blend from and to those points, matching the general trajectory shape of the non-primary selection to fill the gap.

When dealing with marker occlusions, often there is a nearby marker or markers whose transform pattern (generalized trajectory) is sufficiently similar to provide the information needed to fill the gap.

Functional area

Data editing

Command syntax

Syntax

```
copyPattern
```

Arguments

None

Flags

None

Return value

void

Examples

```
// The gap existing in LFWT between frames 100 and 300
// is filled using the key pattern that exists in LMWT
// over the same frame range.
select LMWT LFWT;
selectRange 100 300;
copyPattern;
```

Additional information

Related commands

- [fillGaps \(page 389\)](#)
- [findGap \(page 404\)](#)

copyString

Description

Summary

Copies a string to the clipboard.

Details

Copies a string to the system clipboard, as unformatted, plain text.

Functional area

System

Command syntax

Syntax

```
copyString text
```

Arguments

Name	Type	Required	Comments
text	string	yes	Text to copy to the clipboard.

Flags

None

Return value

void

Examples

```
// Copy some text to the Clipboard
string $str1 = "This is ";
string $str2 = "some text.";
string $result;
$result = $str1 + $str2;
// Copy it
copyString $result;
```

COS

Description

Summary

Returns the cosine of an input angle (degrees)

Details

The cos command is used to calculate the cosine for any input angle. The input units are in degrees.

Picture the unit circle (a circle of radius 1.0 centered about the origin). If you were to imagine a unit vector rotating inside of the unit circle, the angle (in degrees) that the unit vector makes with the positive x-axis is the input argument theta. The x-coordinate of the endpoint of the unit vector is the return value of the cos function.

Functional area

Math

Command syntax

Syntax

```
cos value
```

Arguments

Name	Type	Required	Comments
value	float	yes	Float value of an angle in degrees.

Flags

None

Return value

float

Returns a floating point value in the range [-1.0, 1.0]

Examples

```
float $temp = (cos (30));  
// declares a variable for the cosine function  
  
print (string ($temp));  
// prints the output of the cosine function
```

Additional information

Related commands

- | [acos \(page 103\)](#)
- | [atan \(page 148\)](#)
- | [sin \(page 1199\)](#)
- | [tan \(page 1308\)](#)

cpExportOptions

Description

Summary

Set export options for camera parameterization file

Details

Set export options for camera parameterization file. When this type of the file is saved the options affect the output format

Functional area

File handling

Command syntax

Syntax

```
cpExportOptions [-writeForIQ boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
writeForIQ	1	boolean	—	Save the file in a format suitable for Vicon iQ, which uses the camera numbers rather than the device ID.

Return value

void

Examples

```
// Save camera parameterization file in IQ format
loadFile -createSecondFig -importType "selCreateNew" "C:/Shogun/Thursday July20
/00001/00001.cp";
cpExportOptions -writeForIQ true;
saveFile "C:/IQ/00001.cp";
```

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [mcpImportOptions \(page 798\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dImportOptions \(page 1375\)](#)
- | [xcplImportOptions \(page 1377\)](#)

create

Description

Summary

Create one or more new modules of a specified type.

Details

Use create to define one or more new objects in your scene. You may use this command to create any of the six object types: Markers, RigidBodies, Characters, BoneNodes, and Solvers. Various attributes of the object can be set at creation, including pre translation and pre rotation values, parent module, and the name of the module(s) to create.

Functional area

Data editing

Command syntax

Syntax

```
create moduleType "name1" ["name2"] ...[-addSelect] [-parent string] [-positionOffset vector] [-rotationOffset vector] [-preTranslation vector] [-preRotation vector] [-allClips]
```

Arguments

Name	Type	Required	Comments
moduleName	string	no	String name of the new module
moduleType	string	yes	Type of module to be created. Valid types are: Marker, RigidBody, Character, BoneNode, and Solver.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
copy	0	—	—	Create copies of all selected objects of the named type. Copies will have same name but no other data in common.
addSelect	0	—	—	Do not reset the selection. Simply adds this new object to the current selection.
parent	1	string	—	Parent the newly created object to the object named here.
preTranslation	1	vector	—	Offset the newly created object by posOffsetVector from its parent.
preRotation	1	vector	—	Offset the newly created object by rotOffsetVector from its parent.
allClips	0	—	—	Add the new module to all clips (default is active clip only)
positionOffset	1	vector	—	Sets the pre translation value - deprecated: use -preTranslation instead
rotationOffset	1	vector	—	Sets the pre rotation value - deprecated: use -preRotation instead

Return value

void

Examples

```
// Create a new object of type Marker having the name "rightThumb"  
// parent the new Marker to the CharacterNode called Actor_1.  
create Marker rightThumb -parent Actor_1;  
  
// Create 4 new characters  
create Character Actor_2 Actor_3 Actor_4 Actor_5;
```

Additional information

Related commands

- | [createKey \(page 256\)](#)
- | [delete \(page 322\)](#)
- | [selectByType \(page 975\)](#)

createBoneVirts

Description

Summary

Create virtual markers for all selected bones in the scene, and animate them across the animation range.

Details

Create virtual markers for all selected bones in the scene, and animate them across the animation range. Useful for the creation of nodes that will be used to control an IK rig.

Functional area

Data editing

Command syntax

Syntax

```
createBoneVirts [-joint]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
joint	0	—	—	Use this option to create a virtual marker at the joint's position

Return value

string array

Returns a string array containing the names of the added markers

Examples

```
// load an ASF/AMC pair
// select bones
selectByType BoneNode;
createBoneVirts -joint;
```

createCheckBox

Description

Summary

Create a check box user control on the given user window.

Details

Creates a check box user control on the user window specified by `windowID`. A check box gives the user a yes/no or true/false option. The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command. The command returns the Control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createCheckBox parentWindowID [-hidden] [-text string] [-pos integer
array] [-form integer] [-triState] [-left] [-check]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
triState	0	—	—	Specifies that the check box have an "indeterminate" state, in addition to "checked" and "unchecked".
left	0	—	—	Specifies that the button text appear on the left side of the check box. The default is text on the right.
check	0	—	—	Specifies that the check box is initially "checked". The default is "unchecked".
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
text	1	string	—	The text that gets displayed with the control. The default is no text.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a check box 20 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a Check Box User Control.
int $windowId;
int $controlId;
int $rect[4];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Specify the size of the Control - 100 units wide
$rect[0] = 20; // Left
$rect[1] = 20; // Top
$rect[2] = 120; // Right
$rect[3] = 40; // Bottom

// Create the Control on the Window, passing
// in the Window Id of the User Window we
// just created. Make the Check Box tri-state and set its text
$controlId = `createCheckBox $windowId -triState -text "Check Me!" -pos $rect`;
```

Additional information

Related commands

- [getCheckBoxCheck \(page 465\)](#)
- [setCheckBoxCheck \(page 1033\)](#)

createColorPicker

Description

Summary

Create a color picker user control on the given user window.

Details

Creates a color picker user control on the user window specified by `windowID`. A color picker allows the user to specify a color, using either the predefined colors, or by bringing up the custom color map.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the Control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createColorPicker parentWindowID[-hidden] [-pos integer array] [-form  
integer] [-color integer array]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of User Window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
color	1	integer array	—	A three element int array representing RGB values from 0255, specifying the initial color of the control. The default is blue.
pos	1	integer array	—	A four element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a color picker 20 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a Color Picker User Control.
int $windowId;
int $controlId;
int $color[3];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;
// Specify the initial color of the control, Red
$color [0] = 255; // Red
$color [1] = 0;   // Green
$color [2] = 0;   // Blue
// Create the Control on the Window, passing
// in the Window Id of the User Window we
// just created. Make the Control tri-state and set
// its text
$controlId = `createColorPicker $windowId -color $color`;
```

Additional information

Related commands

- [getColorPickerColor \(page 475\)](#)
- [setColorPickerColor \(page 1038\)](#)

createDir

Description

Summary

Create a file system directory/folder

Details

Create a file system directory. Will create all folders necessary to satisfy the path to the directory.

Note that this command will never fail, but will return false if it wasn't able to create the directory.

Functional area

System

Command syntax

Syntax

```
createDir "folderPath"
```

Arguments

Name	Type	Required	Comments
folderPath	string	yes	Path to the folder you want to create. Be sure to use forward slashes.

Flags

None

Return value

boolean

Returns true if the directory was successfully created, or false if it wasn't.

Examples

```
// Create a folder on your D: drive. The final  
// folder will be called "Path"  
createDir "D:/Some/Long/Folder/Path/";
```

Additional information

Related commands

- | [pathExists \(page 830\)](#)
- | [setDir \(page 1056\)](#)

createDropList

Description

Summary

Create a drop list user control on the given user window.

Details

Creates a drop list user control on the user window specified by windowID.

A drop list enables the user to select from a set of list of items, with the list dynamically displaying when the user clicks an arrow button.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The height of the "edit" portion of the drop list is always fixed, regardless of the height specified using the `-pos` option or `setControlPos` command. The height of the list portion of the drop list is specified by the height of the control.

The command returns the Control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createDropList parentWindowID "itemStr1" "itemStr2"...[-hidden] [-pos  
integer array] [-form integer] [-sort] [-sel integer]
```

Arguments

Name	Type	Required	Comments
item1	string	no	One or more strings are allowed to populate the list portion of the drop list.
windowId	int	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
sort	0	—	—	Forces the control to automatically sort the items in the list. The default is unsorted.
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
sel	1	integer	—	A zero based index of the item to be selected. The default is no selection.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a drop list 100 units high by 100 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a drop list user control.
int $windowId;
int $controlId;

// First create a user window to place the Control on
$windowId = `createWindow "MyWindow"`;
// Create the Control on the Window, passing
// in the Window Id of the user window we
// just created. Make the list sorted, add some items,
// and set the initial selection
$controlId = `createDropList $windowId "Item #3" "Item #1" "Item #2" -sort -sel
2`;
```

Additional information

Related commands

- | [addDropListItem \(page 105\)](#)
- | [deleteAllDropListItems \(page 324\)](#)
- | [deleteDropListItem \(page 336\)](#)
- | [findDropListItem \(page 401\)](#)
- | [getDropListItem \(page 510\)](#)
- | [getDropListSelectedItem \(page 513\)](#)

createForm

Description

Summary

Create a form user control on the given user window.

Details

Creates a form user control on the given user window specified by `windowID`.

A form is an invisible rectangle that manages the size and position of other user controls. To have controls be managed by a form, use the `-form` option on the control creation command (e.g. [createTextBox \(page 305\)](#)). Then use [setControlAnchor \(page 1045\)](#) to "anchor" the controls to the sides of the form (or to other controls). When the form gets moved or resized, the controls will automatically be re-organized according to the anchor logic specified.

Each user window has a form which can be obtained using [getTopLevelForm \(page 710\)](#). Upon creation of the user window, obtain the top level form, and use it as the argument to the `-form` flag in the control creation commands.

Forms can be nested, forming hierarchies of forms to facilitate complex control layouts. The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the Control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createForm parentWindowID [-hidden] [-pos integer array] [-form integer]
```

Arguments

Name	Type	Required	Comments
windowId	int	yes	ID of user window to place the form on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
pos	1	integer array	—	A four element int array representing a rectangle, which specifies the initial size /position of the form relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a form 200 units high by 200 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this form.
hidden	0	—	—	—

Return value

integer

Examples

```
// Create a Form User Control.
int $windowId;
int $ formId;
int $topForm;
int $controlId;
int $rect[4];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;
// Get the top level form
$topForm = `getTopLevelForm $windowId`;
// Specify the size of the Form - 300x300
```

```
$rect[0] = 20; // Left
$rect[1] = 20; // Top
$rect[2] = 320; // Right
$rect[3] = 320; // Bottom
// Create the Form, adding it as a child of the
// top level form.
$formId = `createForm $windowId -form $topForm -pos $rect`;
// Now create a Text Box, and anchor it so that the text box spans
// the length of the Form
$controlId = `createTextBox $windowId -text "Text Box" -form $formId`;
// Anchor it
setControlAnchor;
$controlId "top" "top" 20;
setControlAnchor;
$controlId "left" "left" 20;
setControlAnchor;
$controlId "right" "right" 20;
// Now, call layoutForm, which will do the managing of the controls
// under the form
layoutForm $topForm;
```

Additional information

Related commands

- [layoutForm \(page 770\)](#)
- [setControlAnchor \(page 1045\)](#)

createGroupBox

Description

Summary

Create a group box user control on the given user window.

Details

Creates a group box user control on the user window specified by `windowID`.

A group box is a form user control with a visual component. A group box is usually used to demarcate groups of controls. As it is a form, it can also manage the layout of those controls. For information on how to use the layout features of a form, see [createForm \(page 250\)](#).

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createGroupBox parentWindowID [-hidden] [-text string] [-pos integer
array] [-form integer]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
text	1	string	—	The text that gets displayed with the control. The default is no text.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the form relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a group box 200 units high by 200 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a group box user control.
int $windowId;
int $groupId;
int $topForm;
int $controlId;

// First create a user window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Get the top level form
$topForm = `getTopLevelForm $windowId`;

// Create the group box, adding it as a child of the
// top level form.
$groupId = `createGroupBox $windowId -form $topForm-text "Group Box"`;
setControlAnchor $groupId "top" "top" 0;
setControlAnchor $groupId "left" "left" 0;
```

```
setControlAnchor $groupId "right" "right" 0;

// Now create a Text Box, and anchor it so that the text box spans
// the length of the group box
$controlId = `createTextBox $windowId -text "Text Box" -form $groupId`;
// Anchor it
setControlAnchor $controlId "top" "top" 20;
setControlAnchor $controlId "left" "left" 20;
setControlAnchor $controlId "right" "right" 20;
// Now, call layoutForm, which will do the managing of the controls
// under the form
layoutForm $topForm;
```

Additional information

Related commands

- [layoutForm \(page 770\)](#)
- [setControlAnchor \(page 1045\)](#)

createKey

Description

Summary

Creates a key on the selected properties of the selected modules at the current frame.

Details

This command will create a keyframe on the selected module. The value set for the key will be a value interpolated from surrounding keys, if any exist. If none exist the values are 0, 0, 0.

Functional area

Data editing

Command syntax

Syntax

```
createKey
```

Arguments

None

Flags

None

Return value

void

Examples

```
// A key is created on the XYZ Translation properties of the markers
// LMWT, and LFWT.
selectProperty "TranslationX" "TranslationY" "TranslationZ";
select LMWT LFWT;
createKey;
```

Additional information

Related commands

- | [copyKeys \(page 224\)](#)
- | [cutKeys \(page 318\)](#)
- | [setProperty \(page 1130\)](#)

createLabelingCluster

Description

Summary

Create a label cluster.

Functional area

Labeling

Command syntax

Syntax

```
createLabelingCluster
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

■ [autoCreateLabelingClusters](#) (page 152)

createListBox

Description

Summary

Create a list box user control on the given user window.

Details

Creates a list box user control on the user window specified by windowId. A list box allows the user to select from a set of list of items.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the Control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createListBox parentWindowID "itemStr1" "itemStr2"...[-hidden] [-pos  
integer array] [-form integer] [-multi] [-sort] [-sel integer]
```

Arguments

Name	Type	Required	Comments
item1	string	no	One or more strings are allowed to populate the list box.
windowId	int	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
multi	0	—	—	Specifies that the control should allow multiple selections. The default is single selection only.
sort	0	—	—	Forces the control to automatically sort the items in the list box. The default is unsorted
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
sel	1	integer	—	A zero based index of the item to be selected. The default is no selection.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a drop list 150 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a list box user control.
int $windowId;
int $controlId;
int $rect[4];

// First create a user window to place the Control on
$windowId = `createWindow "MyWindow"`;
// Specify the size of the Control - 80 units high
$rect[0] = 20; // Left
$rect[1] = 20; // Top
```

```
$rect[2] = 70; // Right
$rect[3] = 100; // Bottom

// Create the Control on the Window, passing
// in the Window Id of the user window we
// just created. Make the list sorted, add some items,
// and set the initial selection. Also allow for multiple
// selection
$controlId = `createListBox $windowId "Item #3" "Item #1" "Item #2" -sort -sel 1
-multi -pos $rect`;
```

Additional information

Related commands

- | [addListBoxItem \(page 110\)](#)
- | [deleteAllListBoxItems \(page 326\)](#)
- | [deleteListBoxItem \(page 340\)](#)
- | [findListBoxItem \(page 407\)](#)
- | [getListBoxItem \(page 578\)](#)
- | [getListBoxSelItems \(page 581\)](#)
- | [selectListBoxItem \(page 989\)](#)

createListView

Description

Summary

Create list view user control

Details

Creates a list view user control. List views are different from list boxes in that they can have multiple columns, check boxes, and grid lines.

Functional area

User Window

Command syntax

Syntax

```
createListView parentWindowID[-hidden] [-pos integer array] [-form  
integer] [-singleSel] [-checkBoxes] [-headerDragDrop] [-gridLines] [-  
editable]
```

Arguments

Name	Type	Required	Comments
parentWindowID	integer	yes	ID of the user window to place the list view on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
hidden	0	—	—	Causes the list view to be initially hidden.
pos	1	integer array	—	A four element integer array holding the rectangular coordinates, from the top left corner of the parent window, of the list view. The order of values should be left, top, right, and bottom.
form	1	integer	—	Adds control as a child of the given form. Forms are used as layout managers for other controls.
singleSel	0	—	—	Makes the list view single-selection. By default, multiple items can be selected.
checkboxes	0	—	—	Adds a check box in the first column of each item in the list view
headerDragDrop	0	—	—	Allows columns in the list view header to be dragged and dropped in the order of the users choosing.
gridLines	0	—	—	Causes lines to be drawn around items in the list view, giving the appearance of a grid.
editable				

Return value

integer

Returns the ID of the list view user control.

Examples

```
// Demonstrate usage of a list view user control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columns
string $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the
// first column. We supply text for subsequent columns using
// setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
```



```
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [deleteListViewItem \(page 342\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

createNumBox

Description

Summary

Create a number box user control on the given user window.

Details

Creates a num box user control on the user window specified by `windowID`.

A num box is the same as text box user control, except it only allows the user to specify a number. A spinner is also available to change the value of the number.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the Control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createNumBox parentWindowID[-hidden] [-pos integer array] [-form integer]
[-flt] [-spin] [-num float]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
flt	0	—	—	Specifies that the number should be evaluated as a floating point number.
spin	0	—	—	Specifies that a spin control be attached to the control.
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
num	1	float	—	Specifies the initial value of the control. Can be either a float or int.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a text box 20 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a Check Box user control.
int $windowId;
int $controlId;
float $value = 5.25;

// First create a user window to place the Control on
$windowId = `createWindow "MyWindow"`;
// Create the Control on the Window, passing
// in the Window Id of the user window we
// just created. Make floating point, specifying the initial
// value and a spinner
$controlId = `createNumBox $windowId -num $value -spin -flt`;
```

Additional information

Related commands

- [getNumBoxNum \(page 610\)](#)
- [setNumBoxNum \(page 1114\)](#)

createPropVST

Description

Summary

Create a prop VST.

Functional area

Labeling

Command syntax

Syntax

```
createPropVST
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

■ [createPropVSTOptions \(page 270\)](#)

createPropVSTOptions

Description

Summary

Sets options for createPropVST command.

Functional area

Labeling

Command syntax

Syntax

```
createPropVSTOptions [-propName string] [-boneColor vector] [-markerColor  
vector] [-stickColor vector] [-autoSaveVsk boolean] [-autoCreateVss  
boolean] [-autoSaveVss boolean] [-reset]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

■ [createPropVST \(page 269\)](#)

createPushButton

Description

Summary

Create a push button user control on the given user window.

Details

Creates a push button user control on the user window specified by `windowId`. A push button is a button that typically fires off an event.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createPushButton parentId [-hidden] [-text string] [-pos integer  
array] [-form integer] [-def]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
def	0	—	—	Specifies that the push button is the "default" button in the user window. When the user hits the Enter key in a user window, the "default" button "push" is automatically generated.
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
text	1	string	—	Specifies the text that appears on the push button. The default is no text.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a Push Button User Control.
int $windowId;
int $controlId;
int $rect[4];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Position the Control
$rect[0] = 20; // Left
$rect[1] = 20; // Top
$rect[2] = 80; // Right
$rect[3] = 45; // Bottom

// Create the Control on the Window, passing
// in the Window Id of the User Window we
// just created. Make it the default button in the User Window
$controlId = `createPushButton $windowId -pos $rect -def -text "Push Me!"`;
```

createRadioButton

Description

Summary

Create a radio button user control on the given user window.

Details

Creates a radio button user control on the user window specified by `windowID`. A radio button is similar to a check box, except that its checked value is mutually exclusive with other radio buttons. Thus, if one radio button gets checked, the other radio buttons in the group automatically uncheck.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createRadioButton parentWindowID [-hidden] [-text string] [-pos integer
array] [-form integer] [-first] [-left] [-check]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
first	0	—	—	Specifies that this radio button is the first in a group of radio buttons. This flag must be set for the radio buttons in a group to work properly.
left	0	—	—	Specifies that the button text appear on the left side of the radio button. The default is text on the right.
check	0	—	—	Specifies that the radio button is initially "checked". The default is "unchecked".
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
text	1	string	—	The text that gets displayed with the control. The default is no text.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a radio button 20 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a Radio Button User Control.
int $windowId;
int $radioId1;
int $radioId2;
int $radioId3;
int $rect[4];

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Specify the size and position of the first Radio Button
$rect[0] = 0; // Left
$rect[1] = 0; // Top
$rect[2] = 100; // Right
$rect[3] = 20; // Bottom

// Create the first Control on the Window. Make sure to
// set the -first flag so that all three can work together
$radioId1 = `createRadioButton $windowId -text "Option 1" -pos $rect -check -
first`;

// Now position the second Radio Button
$rect[0] = 0; // Left
$rect[1] = 25; // Top
$rect[2] = 100; // Right
$rect[3] = 45; // Bottom

// ... and create it
$radioId2 = `createRadioButton $windowId -text "Option 2" -pos $rect`;

// And finally the third Radio Button
$rect[0] = 0; // Left
$rect[1] = 50; // Top
$rect[2] = 100; // Right
$rect[3] = 70; // Bottom
$radioId3 = `createRadioButton $windowId -text "Option 3" -pos $rect`;
```

Additional information

Related commands

- [getRadioButtonCheck \(page 657\)](#)
- [setRadioButtonCheck \(page 1136\)](#)

createReconstructorScript

Description

Summary

Creates a script to the interface or a file that sets the reconstruction options.

Details

Use createReconstructorScript to output a Shogun Post script which saves out the current reconstruction options. The options that are currently set at the time of running the script will be saved.

When the output script is executed, the reconstruction options returns to these values.

The output script can be written to a file, or written to the **Script Editor**. If the command is run with no arguments then the script file will be written to the default path (typically *C:\Users\Public\Documents\Vicon\ShogunPost#. #*).

Functional area

Data editing

Command syntax

Syntax

```
createReconstructorScript [-file string] [-interface]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Writes the script to the given file path.
interface	0	—	—	Writes the created script in the Script Editor panel.

Return value

void

Examples

```
// This command creates a script that saves out
// the current reconstruction settings as an output script
createReconstructorScript -file "c:/reconSettings.hsl";
```

Additional information

Related commands

- [reconstruct \(page 882\)](#)
- [reconstructOptions \(page 887\)](#)

createSceneScript

Description

Summary

Creates a script which can be used to recreate the current scene in Shogun Post.

Details

Use createSceneScript to output a Shogun Post script, saving out the current scene modules and animation.

The generated output script can be run to recreate the exact scene that was saved out, or the script can be edited and used for more generic automatic creation of scene elements. The command can be run with a number of arguments to determine what is contained in the output script. For example, using the `-defaultOnly` flag, modules can be output into the script with no animation.

The output script can be written to a file or to the Script Editor. If the command is run with no arguments then the output script file will be written to the default path. (typically `C:\Users\Public\Documents\Vicon\ShogunPost#. #`).

Functional area

System

Command syntax

Syntax

```
createSceneScript [-file string] [-interface] [-defaultOnly] [-  
curTimeOnly] [-playRangeOnly] [-activeClipOnly] [-selectedOnly] [-  
relativeTime]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Writes the output script to a given file path.
interface	0	—	—	Writes the output script to the Script Editor panel.
defaultOnly	0	—	curTimeOnly, playRangeOnly, activeClipOnly, relativeTime	Creates a script with module information only. No animation data is saved.
curTimeOnly	0	—	defaultOnly, playRangeOnly	Animation data is saved in the script from the current frame only.
playRangeOnly	0	—	curTimeOnly, defaultOnly	Animation data is saved in the script from the play range only. Otherwise data is saved from the animation range.
activeClipOnly	0	—	defaultOnly	Animation data is saved from the active clip only. Otherwise all clips are saved in the script.
selectedOnly	0	—	—	Only selected modules are saved in the script. Animation data for those modules will be saved unless the defaultOnly flag is also used.
relativeTime	0	---	defaultOnly	The output script will ensure that when it is run, any animation data will start at the current frame.

Return value

void

Examples

```
//save out a script to recreate the current Shogun Post scene  
createSceneScript -file "c:/scene1.hsl";
```

```
//save out a script to recreate the current Shogun Post scene  
// with no animation  
createSceneScript -file "c:/scene2.hsl" -defaultOnly;
```

```
//save out a script to recreate the currently selected modules  
//for the play range only  
createSceneScript -file "c:/scene3.hsl" -selectedOnly -playRangeOnly;
```

createShelfButton

Description

Summary

Creates a button on the named tab. If the called tab doesn't exist, it creates a new tab.

Details

Shelf buttons are interface objects that contain script commands. The scripts contained in them may be invoked by pressing the button. Shogun Post saves the current button shelf configuration in a script called *ScriptShelves.hsl*.

Functional area

Interface

Command syntax

Syntax

```
createShelfButton "TabName" [-group string] [-label string] [-script  
string] [-tip string] [-icon string] [-iconIndex integer] [-language  
string] [-update]
```

Arguments

Name	Type	Required	Comments
TabName	string		String name of tab to place button on

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
group	1	string	—	
label	1	string	—	Gives the button this label. Should be 8 characters or less
script	1	string	—	String arguments in your script text should be preceded by the escape character \ as in \"string argument\"
tip	1	string	—	Sets the tip text that will appear when the mouse hovers over the button
icon	1	string	—	Specifies the path to the icon file
iconIndex	1	integer	—	Deprecated. (Icons cannot be picked by index.)
language	1	string	—	Sets whether the script used by the button is Python or HSL.
update	0	—	—	Deprecated. (Shelf now always redrawn).

Return value

void

Examples

```
createShelfButton -label "getBones" -script "string $bones[];
$bones = `getModules -type \"BoneNode\"`;
int $num = `getCount $bones`;
print $num;"-tip "This command will return the number of BoneNodes in the scene";

// This script creates a button called "getBones" and adds it to the
// last button group. When invoked, it returns the number of BoneNodes
```

```
// in the scene. Note that the string after -script is contained within quotes.  
// The \" means that there is a special character. In this case it is a  
// quotation mark \"`.
```

Additional information

Related commands

- | [createShelfGroup \(page 285\)](#)
- | [createShelfTab \(page 289\)](#)

createShelfGroup

Description

Summary

Creates a shelf group.

Functional area

Interface

Command syntax

Syntax

```
createShelfGroup [ "tabName" ] "groupName"
```

Arguments

Name	Type	Required	Comments
tabName	string	no	Optional name of tab
groupName	string	yes	The name of the group to create

Flags

None

Return value

void

Additional information

Related commands

- [deleteShelfGroup \(page 348\)](#)
- [renameShelfGroup \(page 923\)](#)

createShelfSeparator

Description

Summary

Creates a separator at the end of the last button in the specified tab.

Details

Creates a separator at the end of the last button in the specified tab. Useful when initially setting up a tab.

Functional area

Interface

Command syntax

Syntax

```
createShelfSeparator ["tabName"] ["groupName"]
```

Arguments

Name	Type	Required	Comments
tabName			The tab in into which you want the separator inserted.
groupName			

Flags

None

Return value

void

Examples

```
//create a tab and add a separator  
createShelfSeparator "testTab";
```

Additional information

Related commands

- | [createKey \(page 256\)](#)
- | [delete \(page 322\)](#)
- | [selectByType \(page 975\)](#)

createShelfTab

Description

Summary

Creates a shelf tab with the given name.

Details

Shelf buttons are interface objects that contain script commands. Tabs are groups of buttons or layers that may be used to organize the button into user-defined categories.

This command is currently used to build the button shelf when Shogun Post is launched (see *ScriptShelves.hsl* in the main *ShogunPost#.#* directory).

Functional area

Interface

Command syntax

Syntax

```
createShelfTab "tabName1" ["tabName..."]
```

Arguments

Name	Type	Required	Comments
tabName			String representing the name of the tab to create. Quotes around the name are optional although good practice would be to use quotes.

Flags

None

Return value

void

Examples

```
createShelfTab "fileOps"; // This script will create a tab called "fileOps".
```

Additional information

Related commands

■ [createShelfButton \(page 282\)](#)

createSkelScript

Description

Summary

Creates a script that will reproduce a skeleton setup, VST, or VSK.

Details

Use createSkelScript to output a Shogun Post script that may later be used to reproduce the exact replica of a skeletal setup, VST, or VSK.

The command analyzes the bone indicated and all of its descendants. Passing the "root" joint will reproduce the entire skeleton, passing another of the skeleton's bones will reproduce a subset of the skeleton. All of the values of the attributes of the skeleton's bones will be retained.

The skeleton can either be created inside of Shogun Post or imported into Shogun Post from an external source. The functionality of this command can also be executed via the Subject Setup panel.

Use the `-solver` flag when you want to include marker association and solvers into the script.

Functional area

Skeletal solving

Command syntax

Syntax

```
createSkelScript " bonenode" [-file string] [-interface] [-solvers]
```

Arguments

Name	Type	Required	Comments
boneNode			String name of a BoneNode in the scene, generally the root of a skeleton

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Writes the script to the given file path.
interface	0	—	—	Puts the created script in the script editor window
solvers	0	—	—	Writes solver information into the script

Return value

void

Examples

```
// This command will create a script than can build an
// exact replica of the skeleton in the scene.
// This is one of the ways a solving setup can be transferred to
// other motion of the same actor.
createSkelScript -interface root;
```

Additional information

Related commands

■ [create \(page 234\)](#)

createSolvingSetup

Description

Summary

Creates a standard solving setup.

Functional area

Skeletal solving

Command syntax

Syntax

```
createSolvingSetup
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

■ [createSolvingSetupOptions \(page 295\)](#)

createSolvingSetupFromLabelingSetup

Description

Summary

Create solving setup from labeling setup.

Functional area

Skeletal solving

Command syntax

Syntax

```
createSolvingSetupFromLabelingSetup
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

- | [createSolvingSetup](#) (page 293)
- | [createSolvingSetupOptions](#) (page 295)

createSolvingSetupOptions

Description

Summary

Sets options for [createSolvingSetup \(page 293\)](#) command.

Functional area

Skeletal solving

Command syntax

Syntax

```
createSolvingSetupOptions [-boneNamingScheme string] [-fingerSetupType
string] [-lowerArmRollBone boolean] [-upperArmRollBone boolean] [-
lowerLegRollBone boolean] [-upperLegRollBone boolean] [-splitNeckBone
boolean] [-solveMotion boolean] [-boneColor vector] [-autoBoneColor
string] [-boneColorMult float] [-keepBoneLengths boolean] [-
useLabelingSkeletonAsGuide boolean] [-reset]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

■ [createSolvingSetup \(page 293\)](#)

createStaticBox

Description

Summary

Create a static box user control on the given user window.

Details

Creates a static box user control on the user window specified by `windowID`. A static box is simply some text, used to label other controls, or provide feedback.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the Control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createStaticBox parentWindowID[-hidden] [-text string] [-pos integer  
array] [-form integer] [-right]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
right	0	—	—	Specifies that the text be right justified in the control. The default is left justified.
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
text	1	string	—	The text that gets displayed with the control. The default is no text.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a static box 20 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a static box User Control.
int $windowId;
int $controlId;
int $rect[4];

// First create a user window to place the control on
$windowId = `createWindow "MyWindow"`;
// Specify the size of the Control - 100 units wide
$rect[0] = 20; // Left
$rect[1] = 20; // Top
$rect[2] = 120; // Right
$rect[3] = 40; // Bottom

// Create the Control on the Window, passing
// in the Window Id of the User Window we
// just created.
$controlId = `createStaticBox
$windowId-text "This is some text!" -pos $rect`;
```

createTab

Description

Summary

Create a tab controller.

Details

Creates a tab user control on the user window specified by `parentWindowID`. A tab is a way to organize user controls.

To have controls be managed by a tab, use the [createForm \(page 250\)](#) command to create a form for every tab. Then control all of the associated forms' visibility based on the active tab (using [getTabSelectedItem \(page 699\)](#)), and the controls for that tab will follow.

Functional area

User Window

Command syntax

Syntax

```
createTab parentWindowID [-hidden] [-pos integer array] [-form integer]
```

Arguments

Name	Type	Required	Comments
<code>parentWindowID</code>	int	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
hidden	0	—	—	The tab is hidden until changed by a user event.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the form relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a form 200 units high by 200 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this form.

Return value

integer

Examples

```
// Create a Tab control and add two tabs
int $windowId;
int $tabControl;
int $firstTab;
int $secondTab;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;
//create a tab control
$tabControl = `createTab $windowId`;
//add tabs
$firstTab = `addTab $tabControl "First"`;
$secondTab = `addTab $tabControl "Second"`;
```

Additional information

Related commands

- [addTab \(page 127\)](#)
- [deleteAllTabItems \(page 330\)](#)
- [deleteTabItem \(page 352\)](#)
- [findTabItem \(page 416\)](#)
- [getTabItem \(page 697\)](#)
- [getTabSelectedItem \(page 699\)](#)
- [selectTabItem \(page 1007\)](#)
- [setTabHandler \(page 1167\)](#)

createTangents

Description

Summary

Create a tangent between a selected key and the adjacent keys on both sides.

Details

Create a tangent between a selected key and the adjacent keys on both sides, which can then use be used to manipulate the curve between keys.

Tangents are created on all translation curves (x, y, and z) for the selected key even when the key is selected on only one curve.

Functional area

Data manipulators

Command syntax

Syntax

```
createTangents ;
```

Arguments

None

Flags

None

Return value

void

Examples

```
// In this example 3 objects are selected, then a range
// within the playRange, then a subset of the object's properties,
// then selecting the range of keys falling within the
// indicated range on the indicated properties.
// Finally tangents are created on the selected keys.
select LMWT;
selectRange 200 240;
selectProperty Translation;
selectKeys -ranges;
createTangents;
```

Additional information

Related commands

- [breakTangents \(page 177\)](#)
- [deleteTangents \(page 354\)](#)
- [unbreakTangents \(page 1321\)](#)

createTextBox

Description

Summary

Create a static box user control on the given user window.

Details

Creates a static box user control on the user window specified by `windowID`. A static box is simply some text, used to label other controls, or provide feedback.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createTextBox parentWindowID[-hidden] [-text string] [-pos integer array]
[-form integer] [-right] [-center] [-multiLine] [-password] [-readOnly]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
right	0	—	—	Specifies that the text be right justified in the control. The default is left justified.
center	0	—	—	Specifies that the text be centered in the control. The default is left justified.
multiLine	0	—	—	Specifies that text be allowed to span more than one line. This does not cause text to automatically wrap. Rather, hitting Ctrl+Enter will start a new line. Default is single line.
password	0	—	—	Specifies that the text box show *s in place of the text. Good for passwords.
readOnly	0	—	—	Specifies that the text be uneditable. No modification to the text can be made by the user.
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
text	1	string	—	The text that gets displayed with the control. The default is no text.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a text box 20 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a Text Box user control.
int $windowId;
int $controlId;
int $rect[4];

// First create a user window to place the control on
$windowId = `createWindow "MyWindow"`;
// Specify the size of the control - 100 units wide
$rect[0] = 20; // Left
$rect[1] = 20; // Top
$rect[2] = 120; // Right
$rect[3] = 40; // Bottom

// Create the control on the window, passing
// in the Window Id of the user window we just created.
$controlId = `createTextBox $windowId-text "This is some text!" -pos $rect`;
```

Additional information

Related commands

- | [getControlText \(page 493\)](#)
- | [setControlText \(page 1051\)](#)

createTimeBox

Description

Summary

Create a time box user control on the given user window.

Details

Creates a time box user control on the user window specified by `windowID`. A time box is a text box which specifically deals with frame numbers, respecting the time appearance in Shogun.

The control is initially placed in the top-left corner of the user window, and must be placed by using the `-pos` option, or by using the [setControlPos \(page 1048\)](#) command.

The command returns the control ID of the user control, which should be saved for later operations on the control.

Functional area

User Window

Command syntax

Syntax

```
createTimeBox parentWindowID[-hidden] [-pos integer array] [-form integer] [-frame integer]
```

Arguments

Name	Type	Required	Comments
<code>windowId</code>	<code>int</code>	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
hidden	0	—	—	Specifies that the control should be hidden initially. Call showControl (page 1195) to subsequently make it visible. The default is for it to be visible.
frame	1	integer	—	Specifies that the frame number to display.
pos	1	integer array	—	A four-element int array representing a rectangle, which specifies the initial size /position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom. The default is a time box 20 units high by 50 units wide.
form	1	integer	—	Control ID of the form user control which will dynamically position this control.

Return value

integer

Examples

```
// Create a time box user control.
int $windowId;
int $controlId;
int $rect[4];
int $playEnd;

// First create a user window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Specify the size of the Control - 100 units wide
$rect[0] = 20; // Left
$rect[1] = 20; // Top
$rect[2] = 120; // Right
$rect[3] = 40; // Bottom
```

```
// Get the end of the play range
$playEnd = `getPlayEnd`;

// Create the Control on the Window, passing
// in the Window Id of the user window we
// just created. Set the frame number to be
// the end of the play range.
$controlId = `createTimeBox $windowId-pos $rect -frame $playEnd`;
```

Additional information

Related commands

- [getControlText \(page 493\)](#)
- [setControlText \(page 1051\)](#)

createWindow

Description

Summary

Creates a blank docking user window.

Details

Creates a blank docking user window that is similar to the other docking windows found in Shogun Post. This command is the starting point for all custom GUI work inside of Shogun Post.

The window name must not match the window name of any other docking window (user or native), or the command will fail.

The return value is the unique ID of the user window. Save this ID, as it will be used in many of the other custom GUI commands.

Functional area

User Window

Command syntax

Syntax

```
createWindow "windowName"
```

Arguments

Name	Type	Required	Comments
windowName	string	yes	Unique window name. Must not match any other docking window names, user or native.

Flags

None

Return value

integer

Examples

```
// Create a user window and store its user window ID.  
// Save the ID to our ini file so we can retrieve it  
// from other scripts that might need it  
int $windowId;  
  
// Create the user window  
$windowId = `createWindow "MyWindow"`;  
  
// Save the user window ID to our ini, so we can use it later.  
writeProfileInt "MyWindowSection" "MyWindowID" $windowId;
```

Additional information

Related commands

- [destroyWindow \(page 358\)](#)
- [getTopLevelForm \(page 710\)](#)
- [getWindowRect \(page 723\)](#)

cropClip

Description

Summary

Crops the active clip on the left side, right side, or both sides.

Details

cropClip used with no flags crops the active clip to the current play range In and Out. When -curTime is used you must specify either -leftOnly or -rightOnly.

Functional area

Data manipulators

Command syntax

Syntax

```
cropClip [-leftOnly] [-rightOnly] [-curTime]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
leftOnly	0	—	rightOnly	Crops the beginning of the active clip to the start of the play range (or to the current time, if -curTime is specified)
rightOnly	0	—	leftOnly	Crops the beginning of the active clip to the start of the play range (or to the current time, if -curTime is specified)
curTime	0	—	—	Used with either the -leftOnly or rightOnly flags, will use the current time rather than one of the play range ends as the crop time.

Return value

void

Examples

```
// Create a clip and a Marker
create Clip "clip1";

// Set the play range
playRange 20 80;

// Crop the clip
cropClip -leftOnly ;
setTime 50;
cropClip -rightOnly -curTime;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

CROSS

Description

Summary

Returns the cross product of two vectors.

Details

Returns the right handed cross product of vector1 and vector2.

The cross product is the vector that is perpendicular to plane containing the two input vectors.

The cross product is not commutative, meaning that $\text{cross}(\$a, \$b)$ is not equal to $\text{cross}(\$b, \$a)$.

Functional area

Math

Command syntax

Syntax

```
cross vector1 vector2
```

Arguments

Name	Type	Required	Comments
vector1	vector	yes	Left operand of the cross product.
vector2	vector	yes	Right operand of the cross product.

Flags

None

Return value

vector

Returns a vector which is the right handed cross product of vector1 and vector2

Examples

```
vector $a = <<1.0, 0.0, 0.0>>;  
vector $b = <<1.0, 1.0, 0.0>>;  
// show that a x b is not equal to b x a  
print("a x b = " + string(cross($a, $b)));  
print("b x a = " + string(cross($b, $a)));
```

Additional information

Related commands

- | [dot \(page 363\)](#)
- | [getAngleTo \(page 447\)](#)
- | [getLength \(page 576\)](#)
- | [normalize \(page 814\)](#)
- | [setLength \(page 1086\)](#)

cutKeys

Description

Summary

Cuts (deletes) the currently selected keys on the selected nodes.

Details

This command can be used to eliminate a range of problem keys on a group of markers, for instance, or to remove all the animation from the BoneNodes of a skeleton.

Functional area

Data editing

Command syntax

Syntax

```
cutKeys [-all] [-ranges] [-allLayers] [-onMod string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	ranges	Delete all keys on the selected nodes
ranges	0	—	all	Delete all keys within the selected time ranges on the selected nodes.

Name	Flag arguments	Argument type	Exclusive to	Comments
allLayers	0	—	—	Cuts key on all layers. Must specify either -all or -ranges options.
onMod	1	string	—	Cut keys only on specified module.

Return value

void

Examples

```
// These commands select all the BoneNodes in the scene, and then
// remove all the animation from those nodes, resetting the character
// to its base pose.
selectByType BoneNode;
cutKeys -all;

// Cuts any on all selected properties of LFHD within the selected
// time ranges
select LFHD;
cutKeys -ranges;
```

Additional information

Related commands

- | [copyKeys \(page 224\)](#)
- | [copyPattern \(page 226\)](#)
- | [createKey \(page 256\)](#)
- | [delete \(page 322\)](#)
- | [pasteKeys \(page 828\)](#)
- | [setKey \(page 1083\)](#)

dataHealthView

Description

Summary

Enables you to work with data health view properties.

Functional area

Interface

Command syntax

Syntax

```
dataHealthView [-sortBy string] [-sortAscending boolean]  
[viewAnimationRange boolean] [-setShowing string] [-showGaps boolean]
```

Arguments

None

Flags

See above syntax.

Return value

void

dataIssuesMap

Description

Summary

Enables you to work with data issues map properties.

Functional area

Interface

Command syntax

Syntax

```
dataIssuesMap [-enabled boolean] [-subject string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
enabled		boolean		If false, the issues map is disabled. If true, issues map is enabled.
subject		string		String can be subject name or "all" for all subjects. When a subject name is specified, the issues map displays information for the specified subject only.

Return value

void

delete

Description

Summary

Enables you to delete selected modules in a scene.

Details

This command will delete any objects in the scene, including items that do not visually exist in the 3D views (such as constraints and solvers). The command can be used to delete the entire scene, delete selected objects only, or delete only the primary selection of a group of selected objects.

When combined with other commands like those used for determining selected objects based on pre-determined criteria, delete can be used to permanently clear scenes of unnecessary objects.

Functional area

Data editing

Command syntax

Syntax

```
delete [-all] [-primary]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	primary	Deletes all objects
primary	0	—	all	Deletes only the primary object

Return value

void

Examples

```
// Delete all BoneNode objects in the scene
selectByType BoneNode;
delete;
```

deleteAllDropListItems

Description

Summary

Deletes all user drop list items.

Details

Deletes all items in the drop list user control specified by `userControlID`.

Functional area

User Window

Command syntax

Syntax

```
deleteAllDropListItems userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

void

Examples

```
// Delete all the items in a Drop List User Control.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List.
addDropListItem $controlId "Item 1";
addDropListItem $controlId "Item 2";
addDropListItem $controlId "Item 3";

// Delete all the items. The Drop List will be empty after this.
deleteAllDropListItems $controlId;
```

Additional information

Related commands

- | [addDropListItem \(page 105\)](#)
- | [createDropList \(page 247\)](#)
- | [deleteDropListItem \(page 336\)](#)
- | [findDropListItem \(page 401\)](#)
- | [getDropListItem \(page 510\)](#)
- | [getDropListSellItem \(page 513\)](#)
- | [getNumDropListItems \(page 612\)](#)
- | [selectDropListItem \(page 982\)](#)
- | [setDropListHandler \(page 1058\)](#)

deleteAllListBoxItems

Description

Summary

Deletes all user list box items.

Details

Deletes all items in the list box user control specified by `userControlID`.

Functional area

User Window

Command syntax

Syntax

```
deleteAllListBoxItems userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlID</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

void

Examples

```
// Delete all the items in a List Box User Control.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createListBox $windowId`;

// Add some items to the List Box.
addListBoxItem $controlId "Item 1";
addListBoxItem $controlId "Item 2";
addListBoxItem $controlId "Item 3";

// Delete all the items. The List Box will be empty after this.
deleteAllListBoxItems $controlId;
```

Additional information

Related commands

- | [addListBoxItem \(page 110\)](#)
- | [createListBox \(page 259\)](#)
- | [deleteListBoxItem \(page 340\)](#)
- | [findListBoxItem \(page 407\)](#)
- | [getListBoxItem \(page 578\)](#)
- | [getListBoxSellItems \(page 581\)](#)
- | [getNumListBoxItems \(page 616\)](#)
- | [selectListBoxItem \(page 989\)](#)
- | [setListBoxHandler \(page 1088\)](#)

deleteAllListViewItems

Description

Summary

Deletes all items from a list view user control.

Details

Deletes all items from a list view user control.

Functional area

User Window

Command syntax

Syntax

```
deleteAllListViewItems userControlID
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of list view to operate on

Flags

None

Return value

void

Examples

```
// Clear all items from a List View we created earlier.  
int $listViewID = `getGlobalIntVar "MyListViewID"`;  
deleteAllListViewItems $listViewID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteListViewItem \(page 342\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSellItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

deleteAllTabItems

Description

Summary

Delete all tabs associated with the specified `userControlID`.

Details

Delete all tabs associated with the specified `userControlID`.

Functional area

User Window

Command syntax

Syntax

```
deleteAllTabItems userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to be deleted.

Flags

None

Return value

void

Examples

```
// Create a Tab and add an additional tab to the Main Tab.
int $windowId;
int $mainTab;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

//create a main tab controller then add a second tab the main tab
$mainTab = `createTab $windowId "Main" `;
int $secondTab = `addTab $mainTab "Second"`;

//delete the tabs
deleteAllTabItems $mainTab;
```

Additional information

Related commands

- | [addTab \(page 127\)](#)
- | [createTab \(page 300\)](#)
- | [deleteTabItem \(page 352\)](#)
- | [findTabItem \(page 416\)](#)
- | [getTabItem \(page 697\)](#)
- | [getTabSelectedItem \(page 699\)](#)
- | [selectTabItem \(page 1007\)](#)
- | [setTabHandler \(page 1167\)](#)

deleteCharacters

Description

Summary

Deletes all or selected characters in the scene

Details

Deletes all or selected characters in the scene. Note that the whole character is deleted (i.e. all children), not just the character node.

Functional area

Data editing

Command syntax

Syntax

```
deleteCharacters [-all] [-keepLabels]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	—	Deletes all characters. Defaults to deleting selected characters.
keepLabels				

Return value

void

Examples

```
// Delete all Characters in the scene  
deleteCharacters -all;
```

deleteClipData

Description

Summary

Clears all animation data from currently selected module(s) for the active clip

Details

Clears all animation data from currently selected module(s) for the active clip. Useful for cutting all animation data on a module without having to activate different channels

Functional area

Data manipulators

Command syntax

Syntax

```
deleteClipData
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Delete all animation data from Marker1  
select "marker1";  
deleteClipData;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

deleteDropListItem

Description

Summary

Deletes a user drop list item.

Details

Deletes an item in the drop list user control specified by `userControlID`. To delete an item, either provide the index of the item to delete, or a string of the item to delete (but not both). The string must exactly match for it to be deleted.

Functional area

User Window

Command syntax

Syntax

```
deleteDropListItem userControlID itemIndex or "itemString"
```

Arguments

Name	Type	Required	Comments
itemString	int	yes	Matching string of the item to delete. Either an index should be provided, or a string, but not both.
index	int	yes	Zero based index of the item to delete. Either an index should be provided, or a string, but not both.
userControlId	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Delete an item in a Drop List User Control.
int $windowId;
int $controlId;
// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;
// Create the User Control on the Window.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List.
addDropListItem $controlId "Item 1";
addDropListItem $controlId "Item 2";
addDropListItem $controlId "Item 3";
addDropListItem $controlId "Item 4";

// Delete the second and third items.
deleteDropListItem $controlId 1;
deleteDropListItem $controlId "Item 3";
```

Additional information

Related commands

- | [addDropListItem \(page 105\)](#)
- | [createDropList \(page 247\)](#)
- | [deleteAllDropListItems \(page 324\)](#)
- | [findDropListItem \(page 401\)](#)
- | [getDropListItem \(page 510\)](#)
- | [getDropListSelectedItem \(page 513\)](#)
- | [getNumDropListItems \(page 612\)](#)
- | [selectDropListItem \(page 982\)](#)
- | [setDropListHandler \(page 1058\)](#)

deleteFile

Description

Summary

Delete a file from the file system.

Details

Delete a file from the file system. Paths should use forward slashes instead of back-slashes.

If the file cannot be deleted for some reason, the command fails. Note that the file is not moved to the recycle bin, but is permanently deleted.

Functional area

System

Command syntax

Syntax

```
deleteFile "filePath"
```

Arguments

Name	Type	Required	Comments
filePath	string	yes	Full path to the file you want to delete. Be sure to use forward slashes.

Flags

None

Return value

void

Examples

```
// Delete a c3d file from the disk  
deleteFile "D:/SomeProduct/CaptureDay1/Session1/00001.c3d";
```

deleteListBoxItem

Description

Summary

Deletes a user list box item.

Details

Deletes an item in the list box user control specified by `userControlID`. To delete an item, either provide the index of the item to delete, or a string of the item to delete (but not both). The string must exactly match for it to be deleted.

Functional area

User Window

Command syntax

Syntax

```
deleteListBoxItem userControlID itemIndex or "itemString"
```

Arguments

Name	Type	Required	Comments
itemString	int	yes	Matching string of the item to delete. Either an index should be provided, or a string, but not both.
index	int	yes	Zero based index of the item to delete. Either an index should be provided, or a string, but not both.
userControlId	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Delete an item in a List Box User Control.
int $windowId;
int $controlId;
// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;
// Create the User Control on the Window.
$controlId = `createListBox $windowId`;

// Add some items to the List Box.
addListBoxItem $controlId "Item 1";
addListBoxItem $controlId "Item 2";
addListBoxItem $controlId "Item 3";
addListBoxItem $controlId "Item 4";

// Delete the second and third items.
deleteListBoxItem $controlId 1;
deleteListBoxItem $controlId "Item 3";
```

Additional information

Related commands

- | [addListBoxItem \(page 110\)](#)
- | [createListBox \(page 259\)](#)
- | [deleteAllListBoxItems \(page 326\)](#)
- | [findListBoxItem \(page 407\)](#)
- | [getListBoxItem \(page 578\)](#)
- | [getListBoxSellItems \(page 581\)](#)
- | [getNumListBoxItems \(page 616\)](#)
- | [selectListBoxItem \(page 989\)](#)
- | [setListBoxHandler \(page 1088\)](#)

deleteListViewItem

Description

Summary

Deletes an item from a list view user control

Details

Deletes an item from a list view user control

Functional area

User Window

Command syntax

Syntax

```
deleteListViewItem userControlID itemIndex or "itemString"
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of user control item to operate on
itemIndex	any	yes	Value can either be a string or an integer. If it is a string, all items matching the string value will be removed (i.e. multiple items may be removed). If the value is an integer, the item at the (zero-based) index will be removed. If the index is out of range (i.e. below zero or more than the number of items in the list view, the command will fail.

Flags

None

Return value

integer

Returns the number of items left in the list view user control

Examples

```
// Deletes some items from a List View we created earlier.  
int $listViewID = `getGlobalIntVar "MyListViewID"`;  
  
// Deletes the 4th item (indices are zero-based)  
deleteListViewItem $listViewID 3;  
  
// Deletes all items whose first column value is "Mary"  
deleteListViewItem $listViewID "Mary";
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSellItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

deleteRedundant

Description

Summary

Deletes redundant markers from the list of selected markers.

Details

Optical motion capture systems frequently generate redundant trajectories, particularly in cases where the markers to be tracked are near the physical edges of the capture volume. The deleteRedundant command allows you to automatically identify and remove these artifacts by comparing markers with similar names (e.g. LFHD, LFHD_1, LFHD_2, etc).

However, this command will not remove data that is unique and, therefore, possibly useful. If the -tolerance flag is not used, the default setting is 20mm. This operation is applied to the set of selected markers.

Functional area

Data editing

Command syntax

Syntax

```
deleteRedundant [-tolerance float] [-minKeys integer]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
tolerance	1	float	—	The tolerance within which a marker can overlap with another and still have keys cut from it
minKeys	1	integer	—	Minimum length of keys in the fragment required to trigger deletion

Return value

void

Examples

```
// Removes redundant trajectories in the scene.
deleteRedundant;
```

Additional information

Related commands

- [delete \(page 322\)](#)
- [findBadData \(page 395\)](#)

deleteScriptPath

Description

Summary

Remove a directory from the list of script directories that Shogun Post searches for scripts and pipelines.

Details

Removes a directory from the list of script directories searched when Shogun Post builds its internal list of script and pipelines.

By default, this command does not cause the internal list to get rebuilt. To force an immediate rebuild, specify the `-reparse` flag.

If the path specified does not exist in Shogun Post's script directory list, then the command has no effect.

Functional area

System

Command syntax

Syntax

```
deleteScriptPath ScriptPath [-reparse]
```

Arguments

Name	Type	Required	Comments
ScriptPath	string	yes	The full path to the script directory to remove from the script directory list. Be sure to use forward slashes.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
reparse	0	—	—	Reparses the script directories immediately. If not specified, then the script directories will only be reparsed on next Shogun Post launch.

Return value

void

Examples

```
// Remove the given script path and immediately reparse the script
// directories
deleteScriptPath "C:/Program Files/Vicon/ShogunPost#.#/MyScripts/OldScripts/";
```

Additional information

Related commands

- | [addScript \(page 121\)](#)
- | [addScriptPath \(page 123\)](#)
- | [clearScriptPaths \(page 208\)](#)
- | [getScriptPaths \(page 670\)](#)
- | [scriptExists \(page 952\)](#)
- | [setScriptPaths \(page 1151\)](#)

deleteShelfGroup

Description

Summary

Deletes the indicated shelf group.

Functional area

Interface

Command syntax

Syntax

```
deleteShelfGroup ["tabName"] "groupName" [-all]
```

Arguments

Name	Type	Required	Comments
tabName	string	no	Optional string name of the tab to be deleted.
groupName	string	yes	String name of the shelf group to be deleted.

Flags

See above syntax.

Return value

void

Additional information

Related commands

- [createShelfGroup \(page 285\)](#)
- [renameShelfGroup \(page 923\)](#)

deleteShelfTab

Description

Summary

Deletes the indicated tabs (and the buttons on those tabs).

Details

Use deleteShelfTab when you need to delete a shelf tab and all its component buttons. Indicate tabs by specifying the names of the tabs, or the tab indexes, where index ≥ 0 and index $< \text{NumTabsInShelf}$.



Caution

Because you are changing Shogun Post's interface as opposed to changing any data, this operation is not undo-able. To be safe, back up *scriptShelves.hs/* before running this command.

Functional area

Interface

Command syntax

Syntax

```
deleteShelfTab "tabNameOrIndex1" ["tabNameOrIndex2..."][-all]
```

Arguments

Name	Type	Required	Comments
Index#			An integer indicating the index # of the tab to be deleted; more than one Index # may be invoked for deletion
tabName			The string name of the shelf tab to be deleted; more than one tabName may be invoked for deletion

Flags

See above syntax.

Return value

void

Examples

```
deleteShelfTab 1;
// Deletes the indicated shelf tab(s) and its button contents.
```

Additional information

Related commands

- [createShelfButton \(page 282\)](#)
- [createShelfTab \(page 289\)](#)

deleteTabItem

Description

Summary

Delete a specified tab.

Details

Delete the specified tab associated with the specified `userControlID`.

Functional area

User Window

Command syntax

Syntax

```
deleteTabItem userControlID itemIndex or "itemString"
```

Arguments

Name	Type	Required	Comments
<code>itemString</code>	string	or	The name of the specified tab
<code>itemIndex</code>	int	or	The item index of the specified tab
<code>userControlId</code>	int	yes	ID of user control to be deleted.

Flags

None

Return value

boolean

Examples

```
// Create a Tab and add and additional tab to the Main Tab.
int $windowId;
int $mainTab;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

//create a main tab controller then add a second tab the main tab
$mainTab = `createTab $windowId "Main" `;
int $secondTab = `addTab $mainTab "Second"`;

//delete the Second Tab
deleteTabItem $mainTab 1;
```

Additional information

Related commands

- | [addTab \(page 127\)](#)
- | [createTab \(page 300\)](#)
- | [deleteAllTabItems \(page 330\)](#)
- | [findTabItem \(page 416\)](#)
- | [getTabItem \(page 697\)](#)
- | [getTabSelItem \(page 699\)](#)
- | [selectTabItem \(page 1007\)](#)
- | [setTabHandler \(page 1167\)](#)

deleteTangents

Description

Summary

Removes existing tangents between selected key and the adjacent keys on both sides.

Details

Remove any tangents which exist on selected keys.

Tangents are removed on all translation curves (x, y, and z) for the selected key even when the key is selected on only one curve.

Functional area

Data manipulators

Command syntax

Syntax

```
deleteTangents
```

Arguments

None

Flags

None

Return value

void

Examples

```
// In this example 3 objects are selected, then a range
// within the playRange, then a subset of the object's properties,
// then selecting the range of keys falling within the
// indicated range on the indicated properties and. Finally tangents are
// created then deleted on the selected keys.
select LMWT;
selectRange 200 240;
selectProperty Translation;
selectKeys -ranges;
createTangents;
deleteTangents;
```

Additional information

Related commands

- | [breakTangents \(page 177\)](#)
- | [createTangents \(page 303\)](#)
- | [unbreakTangents \(page 1321\)](#)

delGlobalVar

Description

Summary

Delete a previously assigned global variable identified by name.

Details

Shogun Post's scripting language supports global variables that are accessible across scripts within a session of Shogun Post. This command will delete a global variable specified by its name.

Functional area

Data retrieval

Command syntax

Syntax

```
delGlobalVar name
```

Arguments

Name	Type	Required	Comments
name	string	yes	Name of the variable to delete

Flags

None

Return value

void

Examples

```
Script A
// Define a global variable
setGlobalVar myVar 1.1;...
```

```
Script B
// Check to make sure global variable set in Script A still exists
if (`getGlobalVarExists myVar`){
// Use it ...
// Delete the global variable
delGlobalVar myVar;
```

Additional information

Related commands

- | [getGlobalBooleanVar \(page 550\)](#)
- | [getGlobalFloatVar \(page 552\)](#)
- | [getGlobalIntVar \(page 554\)](#)
- | [getGlobalStringVar \(page 556\)](#)
- | [getGlobalVarExists \(page 558\)](#)
- | [getGlobalVectorVar \(page 560\)](#)
- | [setGlobalVar \(page 1074\)](#)

destroyWindow

Description

Summary

Destroys an existing docking user window.

Details

Destroys a docking user window, which was previously created by calling [createWindow \(page 311\)](#).

When a user window is destroyed, all of its forms and controls are destroyed along with it. Thus, only the user window itself needs to be destroyed.

Upon shut down, Shogun Post automatically destroys all user windows.

Functional area

User Window

Command syntax

Syntax

```
destroyWindow userWindowID
```

Arguments

Name	Type	Required	Comments
windowId	int	yes	The user window ID previously obtained from calling createWindow (page 311) .

Flags

None

Return value

void

Examples

```
// Create and Destroy a user window
int $windowId;
// Create the user window
$windowId = `createWindow "MyWindow"`;

// ... and immediately destroy it
destroyWindow $windowId;
```

dirChooser

Description

Shows the system File Browser dialog box, which allows users to choose a folder.

Functional area

User Window

Command syntax

Syntax

```
dirChooser [-owner integer] [-title string] [-path string] [-failOnCancel]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
owner	1	integer	—	—
title	1	string	—	—
path	1	string	—	—
failOnCancel	0	—	—	—

Return value

string

dockWindow

Description

Summary

Docks a panel docking window to the side of the main Shogun Post window.

Details

This command gives the user the ability to programmatically dock a panel docking window to the side of the main Shogun Post window, or to the side of another window. Usually, the user does this by clicking and dragging the mouse.

Both Shogun Post's panel docking windows and user windows can be controlled using this command.

Functional area

Interface

Command syntax

Syntax

```
dockWindow "windowName" "side" [-to string]
```

Arguments

Name	Type	Required	Comments
windowName	string	yes	Name of the window to dock. Can be an existing Shogun Post docking windows or a user window.
side	string	yes	The side of the main application window to dock to. Valid values are l, left, r, right, t, top, b, or bottom. If the -to flag is specified, then it will dock to the side of the other window

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
to	1	string	—	Allows you to dock the window to the side of another docking window. Using this flag allows you to create complex window layouts

Return value

void

Examples

```
// Create a User Window
createWindow "DockTest";

// Dock it to the left of the Attributes window
dockWindow "DockTest" "left" -to "Attributes";
```

Additional information

Related commands

- | [autohideWindow \(page 156\)](#)
- | [floatWindow \(page 428\)](#)
- | [setWindowSize \(page 1191\)](#)
- | [showWindow \(page 1197\)](#)
- | [tabWindow \(page 1306\)](#)

dot

Description

Summary

Returns the dot product of the indicated vectors

Details

Use dot when you need to obtain the dot product of two vectors.

The dot product of two vectors, v1 and v2 is defined to be $v1.x*v2.x + v1.y*v2.y + v1.z*v2.z$. The dot product is also equal to the length of v1 times the length of v2 times the cosine of the angle between them.

While the dot function is available, the Shogun Post function `getAngleTo` should be used when computing angles between vectors.

Functional area

Math

Command syntax

Syntax

```
dot vector1 vector2
```

Arguments

Name	Type	Required	Comments
v1	vector	yes	Vector variable
v2	vector	yes	Vector variable

Flags

None

Return value

float

Returns the dot product of v1 and v2

Examples

```
vector $v1 = <<1, 0, 0>>;
vector $v2 = <<0, 1, 0>>;

// show that the dot product between two orthogonal vectors is = 0
print( string( dot($v1, $v2) ) );

// show that the formula below is equivalent (for these particular $v1
// and $v2 choices) to a call to getAngleTo
print( acos( dot($v1, $v2) / (getLength($v1)*getLength($v2)) ) );
print(getAngleTo($v1, $v2)) ;
```

Additional information

Related commands

- | [getAngleTo \(page 447\)](#)
- | [getLength \(page 576\)](#)
- | [setLength \(page 1086\)](#)

enableControl

Description

Summary

Enable/disable a user control.

Details

Enables or disables the user control specified by `userControlId`.

A control that is disabled is blocked from user input (mouse and keystrokes). It is also dimmed to indicate a disabled status.

Disable a control when you want to temporarily block users from interacting with a control.

All user controls are enabled by default.

Functional area

User Window

Command syntax

Syntax

```
enableControl userControlID enableBoolean
```

Arguments

Name	Type	Required	Comments
enableBoolean	boolean	yes	Flag to enable/disable the user control
userControlId	int	yes	ID of user control to enable/disable.

Flags

None

Return value

boolean

Examples

```
// Show how to disable a control.
int $windowId;
int $controlId;
boolean $wasEnabled;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Push Button Control in the window. We will disable it
// after we create it.
$controlId = `createPushButton $windowId -text "Disabled"`;

// Now disable the control
$wasEnabled = `enableControl $controlId false`;
print $wasEnabled;
```

Additional information

Related commands

- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlText \(page 1051\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)
- | [showControl \(page 1195\)](#)

exists

Description

Summary

Returns a Boolean of whether or not a module with a name that matches the specified string value exists in the scene.

Details

The exists command is used to query the scene for the existence of a given module or module type. For instance, you may be writing an automatic editing script and are trying to determine whether or not a certain marker, like LUPA (left upper arm) exists or not, in order to determine what method to use for filling in gaps on other arm related markers.

Functional area

Data editing

Command syntax

Syntax

```
exists modPath [-type string] [-priorityOnly]
```

Arguments

Name	Type	Required	Comments
modPath	string	yes	Name of module to be queried

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
type	1	string	—	Type of module for which to check existence
priorityOnly	0	—	—	Search only the priority module

Return value

boolean

Examples

```
// either one returns a value of True if this node exists in the scene
boolean $t = `exists "Actor_1"`;
boolean $v = exists( "Actor_1" );
```

Additional information

Related commands

- [hasKey \(page 734\)](#)
- [isSelected \(page 755\)](#)

exit

Description

Summary

Quits the Shogun Post application

Details

Will cause Shogun Post to begin its shut down process. You are prompted to save any unsaved data before completely shutting down.

Functional area

System

Command syntax

Syntax

```
exit
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Perform some long operation and save the scene
// ...
saveFile "C:/data/myfile.hdf";

// Shut down Shogun Post. Since we just saved the scene, we won't be
// prompted to save the scene before shutting down.
exit;
```

Additional information

Related commands

- [loadFile \(page 779\)](#)
- [newFile \(page 812\)](#)

extrapolateFingers

Description

Summary

Extrapolate first joint finger data to second and third joint.

Functional area

Skeletal solving

Command syntax

Syntax

```
extrapolateFingers [-ranges] [-currentFrame] [-selectedCharacters] [-  
activeCharacters] [-multiplier float]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

- | [getExtrapolateFingersAfterSolve \(page 525\)](#)
- | [setExtrapolateFingersAfterSolve \(page 1067\)](#)

fabs

Description

Summary

Returns the floating point absolute value of a number.

Details

The fabs command will return the absolute value of a floating point number. This return value will always be greater than or equal to zero.

If a variable of type int is passed in as an argument, the fabs command will perform a floating point conversion, then return the absolute value of that float.

If the input data is of type int and the output type is desired to be of type int, use the [abs \(page 101\)](#) command.

Functional area

Math

Command syntax

Syntax

```
fabs number
```

Arguments

Name	Type	Required	Comments
number	float	yes	If number is of type int, a floating point conversion will be performed.

Flags

None

Return value

float

Returns a floating point value which is greater than or equal to 0

Examples

```
// Compute some absolute values
float $absVal;
$absVal = `fabs -5`;
print $absVal; // Should be 5.000000
$absVal = `fabs -10.4520`;
print $absVal; // Should be 10.452000
```

Additional information

Related commands

■ [abs \(page 101\)](#)

fastForward

Description

Summary

Go to the end of the play range.

Details

fastForward snaps the timeline to the last frame in the current play range.

Functional area

Playback control

Command syntax

Syntax

```
fastForward
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

- [play \(page 832\)](#)
- [rewind \(page 935\)](#)

fbxExportOptions

Description

Summary

Sets export options for the FBX exporter.

Details

Sets export options for the FBX exporter for the next time you export an FBX.

Functional area

File handling

Command syntax

Syntax

```
fbxExportOptions [-collapseSubjects boolean] [-exportActiveClipOnly  
boolean] [-convertZup boolean] [-exportLabelingSetups boolean] [-  
namespaceMode string] [-unrollAnimationCurves boolean] [-  
exportPlayRangeOnly boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
collapseSubjects	1	boolean	—	Add a root null node named (Actor_1 ... Actor_n) to all root level nodes which are not character nodes.
exportActiveClip Only	1	boolean	—	Export all clips as takes instead of just the current active clip.
convertZup	1	boolean	—	
exportLabelingSe tups	1	boolean	—	
namespaceMode	UseName	string	—	<p>Can be one of:</p> <ul style="list-style-type: none"> None: Removes any present namespace(s) from the module names on export. For example, a module named John:LFHD is exported as LFHD. Warns you if stripping the namespace(s) results in multiple modules having the same name and that the importing application may add <code>_#</code> to resolve that. Short: Creates namespaces in the exported FBX based on the minimum path required to uniquely resolve the module, based on the hierarchy. If used with modules that contain colon-based namespaces, colon-based namespaces are maintained. Generally mixing colon and hierarchy based namespaces in FBX export is something you'd be unlikely to do in practice.

Name	Flag arguments	Argument type	Exclusive to	Comments
				<p>■ Full: Creates namespaces in the exported FBX based on the full path of the module based on the hierarchy. If used with modules that contain colon-based namespaces, colon-based namespaces are maintained. Generally mixing colon and hierarchy based namespaces in FBX export is something you'd be unlikely to do in practice.</p> <p>■ UseName: Exports the modules as named, maintaining colon namespace(s) if present. Warns you if stripping the namespace(s) results in multiple modules having the same name and that the importing application may add <code>_#</code> to resolve that.</p>
<code>unrollAnimationCurves</code>	1	boolean	—	This option handles FBX exported from Shogun that contains Euler or gimbal flipping that was not visible in Shogun. When selected, behavior is similar to that of the Autodesk® MotionBuilder® Gimbal Killer filter on all rotation curves.
<code>exportPlayRangeOnly</code>	1	boolean	—	Exports data from playRange only; any other data is ignored

Return value

void

Examples

```
// Add a root null node named ( Actor_1 ... Actor_n) to all
// root level nodes that are not character nodes.
fbxExportOptions -collapseSubjects off

// Export all clips as take instead of just the current active clip.
fbxExportOptions -exportActiveClipOnly on;
```

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [mcplImportOptions \(page 798\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dImportOptions \(page 1375\)](#)
- | [xcplImportOptions \(page 1377\)](#)

fbxImportOptions

Description

Summary

Set FBX import options for the next time you import an FBX file.

Functional area

File handling

Command syntax

Syntax

```
fbxImportOptions [-stripCharacterNameFromChildren boolean] [-
removeNamespaces boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
stripCharacterNameFromChildren	1	boolean	—	Remove the namespace name from the FBX node name, eg Actor_1:pelvis becomes pelvis.
removeNamespaces	1	boolean	—	If not selected, Shogun retains the colons in the FBX module names.

Return value

void

Examples

```
// Remove namespace from node names, for example,  
//actor_1:pelvis becomes pelvis  
fbxImportOptions -stripCharacterNameFromChildren on;
```

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [mcpImportOptions \(page 798\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dlImportOptions \(page 1375\)](#)
- | [xcplImportOptions \(page 1377\)](#)

fileChooser

Description

Summary

Prompts the user to choose a file using the system file browser.

Details

Prompts the user to select a file on the file system in the system **File Browser**.

The **File Browser** is a dialog box designed for easy file/file name selection. By default, the **File Browser** does not filter on file types, however, filters can be provided using the `-filters` flag. The filters string contains a sequence of filter descriptions, followed by the actual filter (eg, `.c3d`). Descriptions and extensions are separated using the `|` character, as are subsequent descriptions and extensions. For example, to filter on `.c3d` and `.csm` files, a filters string will look like this:

```
"Binary Motion File (.c3d)|.c3d|Character Studio (.csm)|.csm"
```

An `All Files (*.*)` will automatically be appended to any filter string specified.

If the user hits the **Cancel** button (i.e. chooses no directory) and `-failOnCancel` is specified, the command will fail, causing the calling script to abort.

If `-failOnCancel` is not specified, the command will return an empty string (`" "`).

Because the file must return multiple file paths if the `-multi` flag is specified, the command must return an array of strings. If `-multi` is not specified, a one element string array will be returned.

Note that in HSL, directories in file paths are separated by forward-slashes (`/`), rather than backslashes (`\`).

Functional area

User Window

Command syntax

Syntax

```
fileChooser [-owner integer] [-ext string] [-file string] [-dir string] [-filters string] [-multi] [-createPrompt] [-mustExist] [-failOnCancel]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
owner	1	integer	—	Specifies the window ID of a user window to be the "owner" of the File Browser . Essentially centers the Browser to the User Window.
ext	1	string	—	The default file extension (e.g. <i>c3d</i>) to append to the selected filename, if it does not have one. The default is no action.
file	1	string	—	—
dir	1	string	—	Specifies the initial directory of the File Browser when it is displayed. The default is the current directory.
filters	1	string	—	Specifies the file type filters. By default, the File Browser displays all files. Files can be filtered down by extension, or even by <code>fileClose</code>
multi	0	—	—	—
createPrompt	0	—	—	—
must exist	0	—	—	—

Name	Flag arguments	Argument type	Exclusive to	Comments
failOnCancel	0	—	—	—
mustExist	0	—	—	—

Return value

string array

Examples

```
// Prompt the user to select file.
string $files[];
string $initialPath = "C:/";
string $filter = "Binary Motion File (*.c3d)|*.c3d|Character Studio (*.csm)|*.csm";

// Specify the initial directory, the filters, and that the file they
// select must exist.
$files = `fileChooser -dir $initialPath -filters $filter -mustExist`;
print $files;
```

Additional information

Related commands

■ [dirChooser \(page 360\)](#)

fileClose

Description

Summary

Closes a file previously opened with the [fileOpen \(page 387\)](#) command.

Details

Use this command to close a file that you have opened using the fileOpen command, after you are done reading from/writing to the file.

Note that many write operations won't get actually written until the file is closed. You should always close your files as loose files can squander your system resources. You may also not see the result of any writeXXXX commands until you close your file. Also, permissions may prevent you from subsequently opening the file.

Functional area

Disk I/O

Command syntax

Syntax

```
fileClose fileID
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen

Flags

None

Return value

void

Examples

```
int $fileID;  
// Obtain file ID earlier do some operation to the file  
// ...  
// Close the file  
fileClose $fileID;
```

Additional information

Related commands

■ [fileOpen \(page 387\)](#)

fileOpen

Description

Summary

Opens a new file for reading and/or writing.

Details

Use this command as the first step in your custom File I/O scripts.

The file must first be opened before you can read or write to it.

If you are going to be reading a file that already exists on the file system, specify Read mode (using "r" as the second argument). Read mode will not allow you to change the contents of the file (i.e. no writeXXXX commands will be allowed). Read mode is typically used when writing your own file importers.

If you are planning on writing to a file on the file system, then use either Write mode (using "w" as the second argument) or Append mode (using "a" as the second argument).

Write mode always erases any existing data in the file and starts writing from the beginning of the file. Use Write mode when writing your own file exporters.

Append mode does not touch existing file data and only allows you to append data to the end of the file. Use Append mode when writing to cumulative log file, for example.

Note that the file you are trying to open must have proper permissions for the type of operation you are doing, otherwise the command will fail. Be sure to close the file using the [fileClose \(page 385\)](#) command when your script is finished.

Functional area

Disk I/O

Command syntax

Syntax

```
fileOpen "fileName" "openMode" [-b]
```

Arguments

Name	Type	Required	Comments
openMode	string		Can be <code>r</code> (Read mode), <code>w</code> (Write mode), or <code>a</code> (Append mode). See <i>Description</i> above for information on each mode.
fileName	string		Full path to a file in your file system (eg <code>c:/temp/myfile.txt</code>)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
b	0	—	—	—

Return value

integer

Examples

```
int $fileID;
string $fileName = "c:/test.txt";
// Open the file and get the file ID, so we can write to the file
// later
$fileID = `fileOpen $fileName "w"`;

// Write a string to the file, passing in the file ID
// as the first argument. Almost all File I/O commands
// take a file ID as the first argument.
writeString $fileID "This is some text";

// Be sure to close the file after you are done
fileClose $fileID;
```

Additional information

Related commands

■ [fileClose \(page 385\)](#)

fillGaps

Description

Summary

fillGaps operates on selected markers to fill all gaps found within the selected input curves. This command has a number of options that are extremely useful for the flexible handling of marker occlusions.

Details

This command contains a collection of tools to address missing translation keyframes of data usually found in raw optical motion capture data. One of the most useful of these is `fillGaps -rigid -all`, which uses knowledge about the relationships of certain markers to one another to intelligently fill in missing translation data. This option is also flexible since the user can apply a rigid fill based on any selected markers in the scene.

Functional area

Data editing

Command syntax

Syntax

```
fillGaps [-all] [-ranges] [-rigid] [-useCurrentFrame] [-primaryOnly] [-maxGapWidth integer] [-labelingConstraints] [-solvingConstraints] [-rangesOnly] [-noWarning] [-minRigidRange integer integer]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	ranges, rangesOnly	Fill all gaps on selected properties of selected nodes.
ranges	0	—	all, rangesOnly	Fill gaps that fall within the selected time ranges.
rigid	0	—	constraints	Uses a rigid body solution applied to all selected markers to fill all gap(s) in the selection. Can often be successfully used on several selected objects that are not very rigid across an entire move. Shogun will warn the user when more than 9 markers are selected due to calculation overhead.
useCurrentFrame	0	—	—	Bases rigid body calculations off of the current frame rather than letting Shogun pick an optimal rigid body definition (Only used in conjunction with the -rigid option).
primaryOnly	0	—	—	Only fills gaps on the primary selection.
maxGapWidth	1	integer	—	Only fill gaps in which the gap duration in terms of consecutive keys is less than this number.
labelingConstraints	0	—	—	Fills gap(s) as dictated by the offset from the BoneNode attached to.
solvingConstraints	0	—	—	Fills gap(s) as dictated by the offset from the BoneNode attached to.

Name	Flag arguments	Argument type	Exclusive to	Comments
rangesOnly	0	—	all, ranges	—
noWarning	0	—	—	Will suppress the warning generated when the number of markers selected for a rigid fill exceeds 9.
minRigidRange	2	integer	—	—

Return value

integer

Examples

```
fillGaps -rigid -all;
// Execution of this command will result in:
// a) the temporary application of a rigid body solution based
// upon the selected markers
// b) the identification of gaps within the selection set, and
// c) the application of data held within the temp markers in the
// rigid body solution to gaps in the original markers across the
// entire play range.
// This command will not overwrite any data pertaining to the selected
// markers, it merely fills in data where it is found to be missing.

fillGaps -maxGapWidth 5;
// Fill all gaps in the selection set that are 4 keys or
// less in duration.
```

Additional information

Related commands

- [copyData \(page 222\)](#)
- [copyPattern \(page 226\)](#)
- [makeRigid \(page 785\)](#)
- [rigidBody \(page 937\)](#)

filter

Description

Summary

The filter command will filter keys across the current playRange on the selected properties of the selected modules using the current filter settings.

Details

Shogun Post's filter command is a carefully crafted, flexible tool designed to address the particular challenges presented by noise commonly found in motion capture system output. The tool uses an adaptive recursive methodology that analyzes keyframes in sequence to isolate digital noise from the sharp translation and rotational transitions found in biomechanical motion. Many filters written for motion capture tools have been incapable of drawing such distinctions automatically, forcing artists into an unhappy choice between time-consuming manual filtering and indiscriminate automatic filtering that destroys the nuance in the performances.

Functional area

Data editing

Command syntax

Syntax

```
filter lightCutoff threshold[-selected] [-length integer] [-transWidth  
float] [-lightCutoff float] [-threshold float] [-weightedAverage] [-width  
integer] [-strength integer]
```


Arguments

The effect of the filter is modified with the (optional) argument settings. You can alternatively use the equivalent flags (see examples below).

Name	Type	Required	Comments
lightCutoff	float		Specifies the amount of variation (in mm) allowed in data. A low value results in less allowed variation and thus more filtering.
threshold	float		Useful range 5 to 40 (approx). Scales the lightCutOff value relative to the velocity of the data being sampled. A lower value increases the lightCutOff more slowly as velocity increases.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selected	0	—	—	Filter is applied only to the selected keys
length	1	integer	—	Specifies the number of keys that are sampled
transWidth	1	float	—	Specifies the transition width
lightCutoff	1	float	—	Specifies the amount of variation (in mm) allowed in data. A low value results in less allowed variation and thus more filtering.
threshold	1	float	—	Scales the lightCutOff value relative to the velocity of the data being sampled. A lower value increases the lightCutOff more slowly as velocity increases. A higher value increases the lightCutOff more quickly as velocity increases.

Name	Flag arguments	Argument type	Exclusive to	Comments
weightedAverage	0	—	—	The filter uses a weighted average algorithm instead of Shogun's standard filter algorithm
width	1	integer	—	Only applies when weightedAverage is used. Specifies the number of keys that are averaged.
strength	1	integer	—	Only applies when weightedAverage is used. Specifies the strength of the filter, for example, 3 = low, 5 = medium, 7 = high, 9 = very high, max is 10.

Return value

void

Examples

```
// The following filter is a good, general purpose filter.
// The selected properties of the selected modules are filtered
// using a cutoff value of .1 and a threshold of 35.
filter .1 35;
```

```
//Alternatively, you could achieve the same result using the
//following options.
filter -lightCutoff .1 -threshold 35;
```

Additional information

Related commands

■ [findBadData \(page 395\)](#)

findBadData

Description

Summary

findBadData automatically detects noise or bad data across selected channels.

Details

findBadData is a sophisticated Shogun Post function that provides automatic noise identification. In automated script functions, you can call out a series of separate findBadData operations, each with varying parameters.

Each operation will in its turn identify different types (higher frequency vs. lower frequency, for instance) of "bad" data that may then be handled separately with the appropriate applications of [filter \(page 392\)](#), [cutKeys \(page 318\)](#), and/or [fillGaps \(page 389\)](#) commands.

The command uses the same data identification routines found in the filter command coupled with a findBadData threshold parameter to adjust the effect. As you decrease or tighten the findBadData threshold, the greater the number of "bad" keys will be selected, all other parameters being equal.

Functional area

Data retrieval

Command syntax

Syntax

```
findBadData [thresholdValue [ lightCutOff [sensitivity]]][-all] [-ranges]  
[-select integer integer]
```

Arguments

Name	Type	Required	Comments
cutoff	Frequency		Useful range 0.01 to 0.45
threshold			This value is expressed in millimeters

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	ranges	Select all keys in the play range that are potentially "bad".
ranges	0	—	all	Select all keys in selected time ranges that are potentially "bad".
select	2	integer	—	Select the indicated [int] number of keys on either side of each bad key found.

Return value

void

Examples

```
select LFWT;
findBadData -all -select 1 1 5 0.1 35;
// This execution of the findBadData command will effectively
// use the filter settings 0.1 [cutoff frequency] and 35 [threshold]
// to identify noisy segments within the entire frame range of the
// translation properties of the LFWT Marker node. Further, a
// findBadData threshold of 5 (a value that expresses a variance
// allowance from the provisionally filtered curve) will refine
// the filter selection. The '-select 1 1' option adds 1 keyframe on
// the front and 1 on the end of each noisy segment that is identified.
// To increase the number of keys selected in this instance, decrease
// the value of the findBadData threshold. Lowering the threshold will
// loosen the tolerance and increasing the value will tighten the
// tolerance.
```

Additional information

Related commands

- [fillGaps \(page 389\)](#)
- [filter \(page 392\)](#)
- [findSelectedKeys \(page 414\)](#)

findDeviantKeys

Description

Summary

Finds and selects keys that deviate over a tolerance between the active clip and another clip.

Details

Finds and selects keys that deviate over a tolerance between the active clip and another clip. Both clips must have a key at each frame for the comparison to be done.

This is useful when comparing the results of an editing operation with the second figure clip.

Functional area

Data retrieval

Command syntax

Syntax

```
findDeviantKeys compareClip tolerance length [-ranges] [-primaryOnly] [-selectRanges] [-noFeedback] [-positionOnly] [-rotationOnly]
```

Arguments

Name	Type	Required	Comments
compareClip	string	yes	Name of the clip to compare with.
tolerance	float	yes	Value in millimeters that key values must be deviant to be selected.
length	integer	yes	Number of consecutive frames that keys must be deviant to be selected.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	0	—	—	Restricts the search to the selected time ranges. The default is to search the entire animation range.
primaryOnly	0	—	—	Restricts the search to the primary selected node. The default is to search all selected nodes.
selectRanges	0	—	—	Specifies that the ranges of the deviant keys should be selected as well.
noFeedback	0	—	—	Set this flag to skip writing the results of the operation to the log
positionOnly	0	—	primaryOnly, selectRanges	Check position deviance only, ignoring rotational deviance
rotationOnly	0	—	ranges, selectRanges	Check rotational deviance only, ignoring positional deviance

Return value

integer

Returns the number of deviant keys

Examples

```
// Import a file and create a second figure
loadFile "C:/temp/myFile.c3d" -i -createSecondFig;

// Select and filter the head markers using the current filter settings
select LFHD RFHD LBHD RBHD;
selectRange -all;filter;

// Now select the keys that deviate from the second figure clip
// by at least 5 mm over at least 10 consecutive keys. Also select the deviant
// ranges.
findDeviantKeys myFilec3d_2fg 5 10 -selectRanges;
// Do something with those keys/ranges!
```

Additional information

Related commands

- [fillGaps \(page 389\)](#)
- [findBadData \(page 395\)](#)
- [findNonRigid \(page 410\)](#)
- [findSelectedKeys \(page 414\)](#)

findDropListItem

Description

Summary

Get the index of a user drop list item.

Details

Searches for a string in the drop list user control specified by `userControlID`. The search is case-insensitive. By default, the search begins at the beginning of the list, searching for `itemString` as an exact match or prefix of any of the drop list items.

If the `-start` flag is specified, the search will begin after the index provided, wrapping back to the top of the list, up until the provided index if a match isn't made.

If the `-exact` flag is specified, the search will only be satisfied if a full string match is made. If `itemString` isn't found, then a value of -1 is returned.

Functional area

User Window

Command syntax

Syntax

```
findDropListItem userControlID "itemString"[-start integer] [-exact]
```

Arguments

Name	Type	Required	Comments
<code>itemString</code>	string	yes	String to find in the drop list.
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
start	1	integer	—	Index of the item to start searching after. By default, the search will begin at the beginning of the list of items.
exact	0	—	—	Performs the search using an exact, case insensitive, string comparison. By default, the search will be satisfied if the search string is a prefix of an item string.

Return value

integer

Examples

```
// Search for a string in a Drop List User Control.
int $windowId;
int $controlId;
int $index;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List.
addDropListItem $controlId "This is text";
addDropListItem $controlId "This";
addDropListItem $controlId "is text";

// First search for the word "this". It will return
// 0, since "this" is a prefix of the first item.
$index = `findDropListItem $controlId "this"`;
print $index;

// Now search for "this" again, this time specifying
// the -exact flag. It will return 1
$index = `findDropListItem $controlId "this" -exact`;
print $index;
```

Additional information

Related commands

- | [addDropListItem \(page 105\)](#)
- | [createDropList \(page 247\)](#)
- | [deleteAllDropListItems \(page 324\)](#)
- | [deleteDropListItem \(page 336\)](#)
- | [getDropListItem \(page 510\)](#)
- | [getDropListSelectedItem \(page 513\)](#)
- | [getNumDropListItems \(page 612\)](#)
- | [selectDropListItem \(page 982\)](#)
- | [setDropListHandler \(page 1058\)](#)

findGap

Description

Summary

This command will identify the next gap found in any of the selected modules.

Details

findGap is useful for quickly identifying gaps in a marker or group of markers. Gaps of one or two frames can be very difficult to detect manually and will introduce problems when the data is applied toward a skeletal solution, or imported into another animation package.

The next marker found using this command becomes the primary-selected object.

Functional area

Data retrieval

Command syntax

Syntax

```
findGap [-selectGapRange integer integer] [-b] [-frameGraph] [-firstFrame] [-any] [-ranges] [-primaryOnly] [-noFeedback] [-rot] [-noLoop]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selectGapRange	2	integer	—	Selects the time range (including leading and trailing frames expressed as integers) of the gap when a gap is found.
b	0	—	—	Searches for gaps backward in time.
frameGraph	0	—	—	Frames the gap in a third the width of the graphView
firstFrame	0	—	—	Goes to the first frame of the gap, in the direction that it is searching
any	0	—	—	Query channels independently. With this option Shogun Post will find gaps on independent channels. E.g. Translation X only. Default is to only find gaps if there are gaps on all three channels.
ranges	0	—	—	Will search selected time ranges, instead of play range
primaryOnly	0	—	—	Causes it to only search for a gap on the primary selected node.
noFeedback	0	—	—	Prevents command from printing to the command log.
rot	0	—	—	Find gaps on rotations channels
noLoop	0	—	—	—

Return value

integer array

Examples

```
findGap -b -select 5 3;
selectKeys -ranges;
// This example selects the time range of the gap immediately
// preceding the current frame, 5 frames before the gap, and
// 3 frames after the gap will be added to the time range.
// Then, all of the keys within the selected time range will also
// be selected.
// Print the gap range of selected Marker
//
//select a Marker
int $i[] = `findGap`;
print $i[0] ;
print $i[1] ;
```

Additional information

Related commands

- [fillGaps \(page 389\)](#)
- [findBadData \(page 395\)](#)

findListBoxItem

Description

Summary

Get the index of a user list box item.

Details

Searches for a string in the list Box user control specified by `userControlID`. The search is case-insensitive.

By default, the search begins at the beginning of the list, searching for `itemString` as an exact match or prefix of any of the list box items.

If the `-start` flag is specified, the search will begin after the index provided, wrapping back to the top of the list, up until the provided index if a match isn't made.

If the `-exact` flag is specified, the search will only be satisfied if a full string match is made. If `itemString` isn't found, then a value of -1 is returned.

Functional area

User Window

Command syntax

Syntax

```
findListBoxItem userControlID "itemString"[-start integer] [-exact]
```

Arguments

Name	Type	Required	Comments
<code>itemString</code>	string	yes	String to find in the list box.
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
start	1	integer	—	Index of the item to start searching after. By default, the search will begin at the beginning of the list of items.
exact	0	—	—	Performs the search using an exact, case insensitive, string comparison. By default, the search will be satisfied if the search string is a prefix of an item string.

Return value

integer

Examples

```
// Search for a string in a List Box User Control.
int $windowId;
int $controlId;
int $index;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createListBox $windowId`;

// Add some items to the List Box.
addListBoxItem $controlId "This is text";
addListBoxItem $controlId "This";
addListBoxItem $controlId "is text";

// First search for the word "this". It will return
// 0, since "this" is a prefix of the first item.
$index = `findListBoxItem $controlId "this"`;
print $index;

// Now search for "this" again, this time specifying
// the -exact flag. It will return 1
$index = `findListBoxItem $controlId "this" -exact`;
print $index;
```

Additional information

Related commands

- | [addListBoxItem \(page 110\)](#)
- | [createListBox \(page 259\)](#)
- | [deleteAllListBoxItems \(page 326\)](#)
- | [deleteListBoxItem \(page 340\)](#)
- | [getListBoxItem \(page 578\)](#)
- | [getListBoxSellItems \(page 581\)](#)
- | [getNumListBoxItems \(page 616\)](#)
- | [selectListBoxItem \(page 989\)](#)
- | [setListBoxHandler \(page 1088\)](#)

findNonRigid

Description

Summary

Assume all selected nodes are intended to be rigidly connected; select any keys on those nodes that stray outside the given rigidity tolerance.

Details

The findNonRigid command is useful for screening out the effects that an errant object motion within a group assumed to be rigid will have on a rigid body applied to those objects.

As a result of digital noise, occlusions, or marker collisions objects that ought to be rigid sometimes display non-rigid behavior; findNonRigid allows you to identify such keys with a tolerance value expressed in millimeters. After identification, you can choose to process or ignore any keys that are found to be non-rigid.

Functional area

Data retrieval

Command syntax

Syntax

```
findNonRigid tolerance[-ranges]
```

Arguments

Name	Type	Required	Comments
tolerance			Float value of the distance tolerance, expressed in millimeters

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	0	—	—	Operates only in selected time ranges.

Return value

integer

Examples

```
select LANK LHEL LMT1 LMT5;
findNonRigid 20;
// Executing these commands will have the effect of selecting all keys
// within the specified left foot markers that are more than 20
// millimeters distant from their rigid location.
// The rigid location for each marker at each frame is that location
// defined by the provisional application of the rigid body.
```

Additional information

Related commands

- [makeRigid \(page 785\)](#)
- [rigidBody \(page 937\)](#)

findPlaneCrossing

Description

Summary

Establish a plane in your scene and find all keys for selected objects that cross that plane

Details

The findPlaneCrossing command is useful for identifying keys that pass through a user-identifiable plane, such as a floor or wall.

The first argument establishes a point on the plane, (i.e. "0 0 50" is a point at 50 millimeters in positive Z).

The second argument is a vector that describes the normal of the plane. For example, "0 0 1" describes a vector with no X or Y direction and a Z component of 1, in other words pointing straight down the world +Z axis. The second argument defines both the orientation and the up side of the plane.

Functional area

Data retrieval

Command syntax

Syntax

```
findPlaneCrossing xPoint yPoint zPoint xDirection yDirection zDirection
```

Arguments

The command requires two arguments to describe the plane: an origin of the plane described in Cartesian terms, and a unit direction vector that defines both the orientation and normal of the plane.

Name	Type	Required	Comments
vector			A direction vector xDirection yDirection zDirection

Name	Type	Required	Comments
z			Location along Global z (float)
y			Location along Global y (float)
x			Location along Global x (float)

Flags

None

Return value

void

Examples

```
select LTOE RTOE;
findPlaneCrossing 0 0 50 0 0 1;
// In this example, findPlaneCrossing selects all keys
// found on the toe markers LTOE and RTOE that penetrate the
// "floor" of the scene, here defined as being all points for
// which the Z component is 50 millimeters. The directional
// arguments 0 0 1 define the vector of the normal for the
// plane, in this case the normal of the plane is aligned with
// world Z, pointing straight "up". If you switch the Z
// component of the direction argument from 1 to -1, the
// command will find all keys that are above the floor.
// This example assumes a Z-up environment.
```

Additional information

Related commands

- [findBadData \(page 395\)](#)
- [findNonRigid \(page 410\)](#)

findSelectedKeys

Description

Summary

Find and move to the next group of selected keys on the currently selected nodes.

Details

The findSelectedKeys command will move the time bar indicator to the beginning frame of the next key selection on the given group of markers (looking forward in time).

In situations where you've selected keys procedurally by using commands such as [findNonRigid \(page 410\)](#) or [findBadData \(page 395\)](#), findSelectedKeys allows you to quickly queue the time bar indicator to the next group of selected keys.

Functional area

Data retrieval

Command syntax

Syntax

```
findSelectedKeys [-b]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
b	0	—	—	Searches backward in time instead of forward in time.

Return value

void

Examples

```
// In this example, all of the keys on the markers LTOE and
// RTOE that are beneath the "floor" (Z values in a Z-up environment
// are less than 100 millimeters) are selected. With findSelectedKeys
// assigned to a hot key, you can quickly cycle through the sets of
// selected keys.
select LTOE RTOE;
findPlaneCrossing 0 0 100 0 0 1;
findSelectedKeys;
```

Additional information

Related commands

- | [findBadData \(page 395\)](#)
- | [findDeviantKeys \(page 398\)](#)
- | [findNonRigid \(page 410\)](#)
- | [findPlaneCrossing \(page 412\)](#)

findTabItem

Description

Summary

Get the index of the specified tab.

Details

Get the index of the specified tab for further manipulation.

Functional area

User Window

Command syntax

Syntax

```
findTabItem userControlID "itemString"[-start integer]
```

Arguments

Name	Type	Required	Comments
itemString	string	no	The name of the tab to be added
userControlId	int	yes	ID of user window to place the control on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
start	1	integer	—	Offset the index from which the command starts

Return value

integer

Examples

```
// Create a Tab and add an additional tab to the Main Tab.
int $windowId;
int $mainTab;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

//create a main tab controller then add a second tab the main tab
$mainTab = `createTab $windowId "Main" "Second" "Third"`;
int $secondTab = `addTab $mainTab "Fourth"`;
int $ThirdTab = `findTabItem $mainTab "Third"`;
print $ThirdTab;
```

Additional information

Related commands

- | [addTab \(page 127\)](#)
- | [createTab \(page 300\)](#)
- | [deleteAllTabItems \(page 330\)](#)
- | [deleteTabItem \(page 352\)](#)
- | [getTabItem \(page 697\)](#)
- | [getTabSellItem \(page 699\)](#)
- | [selectTabItem \(page 1007\)](#)
- | [setTabHandler \(page 1167\)](#)

findTail

Description

Summary

Finds data "tails" near data gaps and selects keys that are members of the tail so you can delete or filter them.

Details

Often gaps are preceded or followed by erratic data segments. As an optical capture system is losing track or re-acquiring a marker, there can be noisy elements on the edges of gaps called tails. The findTails command is designed to identify these unwanted bits of data so that you may remove them.

The `threshold` setting is used to adjust the number of keys selected. The useful range for threshold values is approximately 1-5; 1 selects a lot, and 5 selects very little. After finding tails, the most likely operation is to either filter or cut the select keys.

Functional area

Data retrieval

Command syntax

Syntax

```
findTail threshold
```

Arguments

Name	Type	Required	Comments
threshold	float	yes	Threshold setting is a floating point number greater than 1.0

Flags

None

Return value

void

Examples

```
// This example uses a threshold of 2, which is a common threshold value.  
findTail 2;
```

Additional information

Related commands

- | [cutKeys \(page 318\)](#)
- | [filter \(page 392\)](#)
- | [findBadData \(page 395\)](#)
- | [findDeviantKeys \(page 398\)](#)
- | [findGap \(page 404\)](#)
- | [findNonRigid \(page 410\)](#)
- | [findPlaneCrossing \(page 412\)](#)
- | [findSelectedKeys \(page 414\)](#)

findUnlabeled

Description

Summary

The findUnlabeled command is useful for quickly identifying unlabeled markers.

Details

The findUnlabeled command is designed to identify unlabeled markers. The next marker found using this command becomes the primary selected object. After finding these markers, the most likely operation is to label them.

Functional area

Data retrieval

Command syntax

Syntax

```
findUnlabeled [-b] [-selectRange] [-frameInView] [-continuous]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
b	0	—	—	Searches backward in time instead of forward.
selectRange	0	—	—	Operates only in selected time ranges.

Name	Flag arguments	Argument type	Exclusive to	Comments
frameInView				
continuous				

Return value

void

Examples

```
// In this example, findUnlabeled finds the first ten unlabeled markers
// and prints time range and the name of the markers
int $i;
for( $i = 0;
    $i < 10;
    $i = $i + 1)
{
    findUnlabeled -selectRange;
    int $range[] = `getSelectedTimeRanges`;
    string $sel[] = `getModules -selected`;
    // Range interval and name of the marker
    print $range[0] $range[1] $sel[0];
}
```

Additional information

Related commands

- [findBadData \(page 395\)](#)
- [findDeviantKeys \(page 398\)](#)
- [findGap \(page 404\)](#)
- [findNonRigid \(page 410\)](#)
- [findPlaneCrossing \(page 412\)](#)
- [findSelectedKeys \(page 414\)](#)

fixOcclusion

Description

Summary

Performs occlusion fixing

Details

fixOcclusion performs occlusion fixing using the latest Axiom algorithms. This is the same occlusion fixing that is used by [quickPost \(page 849\)](#), but enables occlusion fixing to be run as a discrete step.

Processes a capture over the entire play range unless the `-ranges` or `-currentFrame` flag is used.

To set the options for occlusion fixing, use the [fixOcclusionOptions \(page 424\)](#) command.

Functional area

Data editing

Command syntax

Syntax

```
fixOcclusion [-ranges] [-currentFrame]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	-	-	currentFrame	Performs occlusion fixing on only the selected range(s) of the current capture using Axiom algorithms.
currentFrame	-	-	ranges	Performs occlusion fixing on only the current frame of the current capture using Axiom algorithms.

Return value

void

Additional information

Related commands

■ [fixOcclusionOptions \(page 424\)](#)

fixOcclusionOptions

Description

Summary

Enables you to specify options for occlusion fixing.

Details

fixOcclusion performs occlusion fixing using the latest Axiom algorithms.

This is the same occlusion fixing that is used by [quickPost \(page 849\)](#), but enables occlusion fixing to be run as a discrete step.

Functional area

Data editing

Command syntax

Syntax

```
fixOcclusionOptions [-fixOcclusionsEnabled boolean] [-applyFixedMarkers
boolean] [-backup boolean] [-markerSmoothing float] [-dataFidelity float]
[-transitionTime float] [-reset] [-fromRTK] [-toRTK]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fixOcclusionsEnabled	1	boolean	—	

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>applyFixedMarkers</code>	1	boolean	—	
<code>backup</code>	1	boolean		If true, a backup of the data is made before occlusion fixing occurs.
<code>markerSmoothing</code>	1	float	—	Higher values produce smoother trajectories.
<code>dataFidelity</code>	1	float	—	Higher values forces the algorithm to follow the reconstructed data more closely.
<code>transitionTime</code>	1	float	—	Transition time in seconds at the boundaries of an occlusion event. Higher values produce smoother trajectories.
<code>reset</code>				Resets all to default values.
<code>fromRTK</code>				
<code>toRTK</code>				

Return value

void

Additional information

Related commands

■ [fixOcclusion \(page 422\)](#)

flatten

Description

Summary

Flatten the active clip.

Details

All layers are flattened to the base layer in the active clip. All the layers are also deleted.

Functional area

NLE

Command syntax

Syntax

```
flatten
```

Arguments

None

Flags

None

Return value

void

Examples

```
flatten;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

floatWindow

Description

Summary

Floats a docking window.

Details

This command gives the user the ability to programmatically "float" a docking window, as well as control its position on the screen. Usually, the user does this by clicking and dragging the mouse.

If neither `-left` nor `-top` are specified, the docking window's position is cascaded in relation to other docking windows.

Both Shogun Post's docking windows and user windows can be controlled using this command.

Functional area

Interface

Command syntax

Syntax

```
floatWindow "windowName" [-left integer] [-top integer] [-width integer]
[-height integer]
```

Arguments

Name	Type	Required	Comments
windowName	string	yes	Name of the window to float. Can be one of Shogun Post's docking windows or a user window.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
left	1	integer	—	Specifies the position, in pixels, of the left side of the docking window, from the top left corner of the screen.
top	1	integer	—	Specifies the position, in pixels, of the top side of the docking window, from the top left corner of the screen.
width	1	integer	—	Specifies the width, in pixels, of the docking window
height	1	integer	—	Specifies the height in pixels, of the docking window

Return value

void

Examples

```
// Create a user window and specify its screen position and size
createWindow "FloatTest";
floatWindow "FloatTest" -left 100 -top 300 -width 300 -height 200;
```

Additional information

Related commands

- | [autohideWindow \(page 156\)](#)
- | [dockWindow \(page 361\)](#)
- | [setWindowSize \(page 1191\)](#)
- | [showWindow \(page 1197\)](#)
- | [tabWindow \(page 1306\)](#)

formatTime

Description

Summary

Converts a system time integer into a human-readable text representation.

Details

Converts a system time integer into a human-readable text representation.

Functional area

System

Command syntax

Syntax

```
formatTime time
```

Arguments

Name	Type	Required	Comments
time	int	yes	System time previously obtained using getSystemTime (page 695) .

Flags

None

Return value

string

Examples

```
// Calculate how long it takes to go through a 1,000,000
// iteration loop. It's about 3 seconds on a 3ghz machine.
int $i, $startTime, $endTime;
string $timeStr;

// Get the start time
$startTime = `getSystemTime`;
$timeStr = `formatTime $startTime`;
print $timeStr;

// Do an empty, lengthy loop
while( $i < 1000000 ){ $i += 1;}

// Get the end time
$endTime = `getSystemTime`;
$timeStr = `formatTime $endTime`;
print $timeStr;

// Now get the execution time
print int( $endTime - $startTime );
```

Additional information

Related commands

- [getSystemTime \(page 695\)](#)
- [system \(page 1303\)](#)

frameToSmpte

Description

Summary

Converts a frame number into a SMPTE timecode string.

Details

Converts a frame number into a SMPTE timecode string, taking into account the SMPTE_Offset value of the active clip.

The SMPTE_Offset value of the active clip specifies the timecode start for the data on the clip. It is there so that frame 1 of your data corresponds to the starting timecode of your take. Without it, your frame numbers would end up large and cumbersome.

If you want to ignore the SMPTE_Offset in the computation, specify the `-ignoreOffset` flag. You may wish to do this when computing a SMPTE_Offset of your own, for instance. In that situation, you wouldn't want the existing offset value in the computation.

The frames (FF) portion of the timecode string will go from 0 to 24, 25, or 30 depending on the current timecode standard (Film, PAL, and NTSC respectively). As your scene frame rate may be a higher multiple of those standard rates (e.g. 120 fps for NTSC), the sub-frame portion indicates the "in between frame" of the frame passed in.

Functional area

System

Command syntax

Syntax

```
frameToSmpte frame[-ignoreOffset]
```


Arguments

Name	Type	Required	Comments
frame	integer	yes	The frame number to convert into SMPTE timecode

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ignoreOffset	0	—	—	By default, the active clip's SMPTE_Offset value is added to the frame argument to produce a final timecode value. This flag causes the offset to be ignored causing a straight frame to timecode conversion.

Return value

string

A SMPTE timecode string in the format HH:MM:SS:FF(SubFrame).

Examples

```
// Print the timecode values of the start and end frames of
// the play range
int $playStart = `getPlayStart`;
int $playEnd = `getPlayEnd`;
string $playStartStr = `frameToSmppte $playStart`;
string $playEndStr = `frameToSmppte $playEnd`;
print $playStartStr;
print $playEndStr;
```

Additional information

Note that, given there are two fields per video frame, as shown in the following example output:

Field 1 subframes are of the format HH:MM:SS:FF(SubFrame), with a colon between the seconds and frames; but

Field 2 subframes are of the format HH:MM:SS.FF(SubFrame), with a period between the seconds and frames.

(Frame rates that use drop frames use a semi-colon (;) and a comma (,) in the equivalent positions.)

For example, with NTSC non-drop timecode at 119.88 fps, the subframes will look like this:

```
00:00:00:00(0) Field 1 of frame 0
00:00:00:00(1) Field 1 of frame 0
00:00:00.00(2) Field 2 of frame 0
00:00:00.00(3) Field 2 of frame 0
00:00:00:01(0) Field 1 of frame 1
00:00:00:01(1) Field 1 of frame 1
00:00:00.01(2) Field 2 of frame 1
00:00:00.01(3) Field 2 of frame 1
Reducing to 59.940 fps gives:
00:00:00:00(0) Field 1 of frame 0
00:00:00.00(1) Field 2 of frame 0
00:00:00:01(0) Field 1 of frame 1
00:00:00.01(1) Field 2 of frame 1
```

Related commands

- [getTime \(page 701\)](#)
- [smpteToFrame \(page 1205\)](#)

getActiveClip

Description

Summary

Return the name of the currently active clip.

Details

Return the name of the currently active clip. Useful when querying the scene for the currently active clip.

Functional area

Data retrieval

Command syntax

Syntax

```
getActiveClip
```

Arguments

None

Flags

None

Return value

string

Examples

```
//create 2 Clips
create Clip "Clip_A" "Clip_B";

//index the Clip
string $clip = `getActiveClip`;

//print the currently active Clip
print $clip;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

getActiveLayer

Description

Summary

Return the name of the active layer of the active clip

Details

Return the name of the active layer of the active clip. An error will be generated if there is no clip on the scene.

Functional area

NLE

Command syntax

Syntax

```
getActiveLayer
```

Arguments

None

Flags

None

Return value

string

The name of the active layer of the active clip

Examples

```
// Print the name of the active clip
string $clip = `getActiveClip`;
print $clip;

// Print the name of the active layer
string $layer = `getActiveLayer`;
print $layer;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

getActiveShelfTab

Description

Summary

Returns the name of the active tab in the shelf.

Details

Returns the name of the shelf tab which is currently selected and active in the GUI.

Functional area

Interface

Command syntax

Syntax

```
getActiveShelfTab
```

Arguments

None

Flags

None

Return value

string

Examples

```
// Print the active tab name to the command line
string $temp = `getActiveShelfTab`;
print $temp;
```

Additional information

Related commands

- | [addTab \(page 127\)](#)
- | [deleteAllTabItems \(page 330\)](#)
- | [deleteTabItem \(page 352\)](#)
- | [findTabItem \(page 416\)](#)
- | [getTabItem \(page 697\)](#)
- | [getTabSellItem \(page 699\)](#)
- | [selectTabItem \(page 1007\)](#)
- | [setTabHandler \(page 1167\)](#)

getActiveTake

Description

Summary

Returns the active take.

Details

Returns the active take. The active take is the current source capture file (x2d file). There can only be one active take and it gets set when you import an x2d file.

When you reconstruct, you will be reconstructing data from the active take.

Use forward-slashes for file paths.

Functional area

System

Command syntax

Syntax

```
getActiveTake
```

Arguments

None

Flags

None

Return value

string

The full path to the active take.

Examples

```
// Imports an x2d file. The active take returned will
// match that of the x2d file imported
loadFile "C:/MyProduct/CaptureDay1/Session1/Take0002.x2d";

// Will print "C:/MyProduct/CaptureDay1/Session1/Take0002.x2d"
print ( `getActiveTake` );
```

Additional information

Related commands

■ [setActiveTake \(page 1023\)](#)

getActiveView

Description

Summary

Returns the index of the active workspace view.

Details

Each workspace view is assigned an index number. If the view is not split, the single view index is 0. If the view is split, index number increases, moving top to bottom then left to right.

This command returns the index number for the selected or active view when run.

Functional area

Interface

Command syntax

Syntax

```
getActiveView
```

Arguments

None

Flags

None

Return value

integer

Examples

```
// Prints the active view index number to the command line
int $temp = `getActiveView`;
print $temp;
```

Additional information

Related commands

- [getActiveViewType \(page 445\)](#)

getActiveViewType

Description

Summary

Returns the type name of the of the active workspace view.

Details

Shogun enables the user to select various types of view, e.g. Perspective, Front, NLE, VPL, Graph, etc. This command returns a string giving the type of view for the currently selected or active view.

Functional area

Interface

Command syntax

Syntax

```
getActiveViewType
```

Arguments

None

Flags

None

Return value

integer

Examples

```
// Prints the name of the active view type to the command line
int $temp = `getActiveViewType`;
print $temp;
```

Additional information

Related commands

- [getActiveView \(page 443\)](#)

getAngleTo

Description

Summary

Returns the angle between two vectors.

Details

Returns the angle between two vectors, in degrees.

Functional area

Math

Command syntax

Syntax

```
getAngleTo vector1 vector2
```

Arguments

Name	Type	Required	Comments
vector1	vector	yes	This is an (unordered) input vector in getAngleTo
vector2	vector	yes	This is an (unordered) input vector in getAngleTo

Flags

None

Return value

float

Returns the absolute value of the angle between vector1 and vector2

Examples

```
vector $v1 = <<1, 0, 0>>;  
vector $v2 = <<0, 1, 0>>;  
// show that getAngleTo returns the absolute value of the angle  
// between two orthogonal vectors (90 degrees) regardless of  
// ordering between vector1 and vector2  
print( getAngleTo($v1, $v2)) ;  
print( getAngleTo($v2, $v1)) ;
```

Additional information

Related commands

- | [cross \(page 316\)](#)
- | [dot \(page 363\)](#)
- | [getLength \(page 576\)](#)
- | [normalize \(page 814\)](#)
- | [setLength \(page 1086\)](#)

getAnimEnd

Description

Summary

Returns the animation end frame.

Details

The getAnimEnd command is used to automatically identify the last frame in the animation range. The value returned may be assigned to an integer variable.

Functional area

Data retrieval

Command syntax

Syntax

```
getAnimEnd
```

Arguments

None

Flags

None

Return value

integer

returns the frame number of the animation Range

Examples

```
int $end = `getAnimEnd`;
print $end;
// In this example, an integer variable named $end is declared and
// assigned the output of the getAnimEnd command.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

getAnimStart

Description

Summary

Returns the animation start frame.

Details

The getAnimStart command is used to automatically identify the first frame in the animation range. The value returned may be assigned to an integer variable.

Functional area

Data retrieval

Command syntax

Syntax

```
getAnimStart
```

Arguments

None

Flags

None

Return value

integer

Returns the Animation start time.

Examples

```
int $start = `getAnimStart`;
print $start;
// In this example, an integer variable named $start is declared and
// assigned the output of the getAnimStart command.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

getAttached

Description

Summary

Retrieves an array of the nodes attached to a bone.

Details

Retrieves an array of the nodes (typically markers) attached to (constraining) a bone.

Functional area

Data retrieval

Command syntax

Syntax

```
getAttached "SolvingBone or LabelingBone Name" [-fullPath] [-nameOnly]
```

Arguments

Name	Type	Required	Comments
SolvingBone or LabelingBone Name	string	yes	BoneNode to retrieve list of attached nodes for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	nameOnly	Specifies that the full path of each node be returned rather than the "minimum" path, which is the minimum path needed to uniquely identify a node.
nameOnly	0	—	fullPath	Specifies that only the name of each node be returned rather than the "minimum" path, which is the minimum path needed to uniquely identify a node.

Return value

string array

Returns a string array with the names/paths of all of the nodes attached to "boneName"

Examples

```
// Get the Markers attached to the head bone.
string $constraints[] = `getAttached "head"`;
print $constraints;
```

Additional information

Related commands

- | [attach \(page 150\)](#)
- | [getAttachedTo \(page 455\)](#)
- | [getConstraintsThisIsSource \(page 483\)](#)
- | [getConstraintsThisIsTarget \(page 485\)](#)
- | [getSolverBones \(page 687\)](#)
- | [setConstraint \(page 1043\)](#)

getAttachedTo

Description

Summary

Retrieves the BoneNode that a node is attached to.

Details

Retrieves the BoneNode that a node is attached to (constraining).

Functional area

Data retrieval

Command syntax

Syntax

```
getAttachedTo "sourceName" [-fullPath] [-nameOnly]
```

Arguments

Name	Type	Required	Comments
sourceName	string	yes	Node to retrieve attached BoneNodes for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	nameOnly	Specifies that the full path of each node be returned rather than the "minimum" path, which is the minimum path needed to uniquely identify a node.
nameOnly	0	—	fullPath	Specifies that only the name of each node be returned rather than the "minimum" path, which is the minimum path needed to uniquely identify a node.

Return value

string array

Returns a string array with the names/paths of all the BoneNodes that are targets in constraints that have the given node as a source.

Examples

```
// Get the BoneNodes the LFHD is attached to.
string $bone[] = `getAttachedTo "LFHD"`;
print $bone;
```

Additional information

Related commands

- | [attach \(page 150\)](#)
- | [getAttached \(page 453\)](#)
- | [getConstraintsThisIsSource \(page 483\)](#)
- | [getConstraintsThisIsTarget \(page 485\)](#)
- | [getSolverBones \(page 687\)](#)
- | [setConstraint \(page 1043\)](#)

getAutoSaveFile

Description

Summary

Get the auto-save file location

Details

Shogun Post has an auto scene-saving feature which will automatically save your scene after a specified period of time since the last file save performed.

By default, the location for the file is a file called *AutoSave.hdf* in your ShogunPost root directory (e.g. *C:/Program Files/Vicon/ShogunPost#.#/AutoSave.hdf*) unless you change it using the [setAutoSaveFile \(page 1027\)](#) command. Use forward slashes instead of back-slashes for all file system paths.

Functional area

System

Command syntax

Syntax

```
getAutoSaveFile
```

Arguments

None

Flags

None

Return value

string

Returns the full path to the auto-save file location.

Examples

```
// Set the auto-save file to be "C:/Temp/AutoSaveFile.hdf"
setAutoSaveFile "C:/Temp/AutoSaveFile.hdf";

// Now print it out - it should match what we just set it to
string $str = `getAutoSaveFile`;
print $str;
```

Additional information

Related commands

- [setAutoSaveFile \(page 1027\)](#)

getBooleanArrayProperty

Description

Summary

Retrieves array of Boolean values from a object's attributes.

Details

Because Shogun Post commands can only have one return type, there must be separate commands for retrieving different types of property data.

This command is mainly used to retrieve the DOF (Degrees of Freedom) attribute. A six element Boolean array is returned. DOFs that are turned on will have "true" array values.

Functional area

Data retrieval

Command syntax

Syntax

```
getBooleanArrayProperty moduleName propertyName
```

Arguments

Name	Type	Required	Comments
propertyName	string	yes	Name of the property on the object.
moduleName	string	yes	Object to retrieve property value from.

Flags

None

Return value

boolean array

Examples

```
// Get the DOFs of the "thorax" bone.  
boolean $dofs[];  
$dofs = `getBooleanArrayProperty "thorax" "DOF"`;  
print $dofs;
```

Additional information

Related commands

- | [getBooleanProperty \(page 461\)](#)
- | [getFloatProperty \(page 543\)](#)
- | [getIntArrayProperty \(page 562\)](#)
- | [getIntProperty \(page 564\)](#)
- | [getProperty \(page 655\)](#)
- | [getStringProperty \(page 693\)](#)
- | [getVectorProperty \(page 717\)](#)
- | [setProperty \(page 1130\)](#)

getBooleanProperty

Description

Summary

Retrieves a Boolean value from an object's attributes.

Details

Because Shogun Post commands can only have one return type, there must be separate commands for retrieving different types of property data.

This command is used to retrieve attributes that can have either an on or off state (e.g. Showing).

It can also be used to retrieve sub-elements of an attribute with many elements (e.g. DOF).

Functional area

Data retrieval

Command syntax

Syntax

```
getBooleanProperty moduleName propertyName[-c] [-d]
```

Arguments

Name	Type	Required	Comments
subPropertyName	string	no	If propertyName represents an array of booleans (e.g. the DOF attribute), this will specify one of the sub-fields.
propertyName	string	yes	Name of the property on the object.
moduleName	string	yes	Object to retrieve property value from.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
c	0	—	—	—
d	0	—	—	—

Return value

boolean

Examples

```
// Get some information about the "thorax" bone.
boolean $hasXRotDof;
boolean $isSpine;
$hasXRotDof = `getBooleanProperty "thorax" "DOF" "Rx"`;
print $hasXRotDof;
$isSpine = `getBooleanProperty "thorax" "Spine_Bone"`;
print $isSpine;
```

Additional information

Related commands

- | [getBooleanArrayProperty \(page 459\)](#)
- | [getFloatProperty \(page 543\)](#)
- | [getIntArrayProperty \(page 562\)](#)
- | [getIntProperty \(page 564\)](#)
- | [getProperty \(page 655\)](#)
- | [getStringProperty \(page 693\)](#)
- | [getVectorProperty \(page 717\)](#)
- | [setProperty \(page 1130\)](#)

getChannels

Description

Summary

Retrieves the selected channels.

Details

Retrieves the list of selected/active channels as an array of channel names.

Some commands require channel names as arguments, and using this command to get the selected channels always ensures that the channels you have selected get operated on, rather than having to hard code the channel names.

Functional area

Data retrieval

Command syntax

Syntax

```
getChannels
```

Arguments

None

Flags

None

Return value

string array

Examples

```
// Set and get active Channels
string $channels[];

// First select some channels
selectProperty TranslationX;
selectProperty TranslationY -a;
selectProperty TranslationZ -a;

// Print out the channel names we just selected
$channels = `getChannels`;
print $channels;
```

Additional information

Related commands

■ [selectProperty \(page 996\)](#)

getCheckBoxCheck

Description

Summary

Get the user check box check value.

Details

Gets the check value of the check box user control specified by `userControlID`.

Because check boxes can be created with the `-triState` flag (the third state being "indeterminate") a Boolean will not be sufficient to accommodate all possible check values. Thus, the command returns an integer: 0 for not checked, 1 for checked, and 2 for "indeterminate".

Functional area

User Window

Command syntax

Syntax

```
getCheckBoxCheck userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Get the checked value of a Check Box User Control
int $windowId;
int $controlId;
int $checked;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createCheckBox $windowId -triState`;

// Get the value on the Check Box.
$checked = `getCheckBoxCheck $controlId`;
if( $checked == 1 )
{
    print "Checked";
}
else if( $checked == 0 )
{
    print "Unchecked";
}
else
{
    print "Indeterminate";
}
```

Additional information

Related commands

- [createCheckBox \(page 239\)](#)
- [setCheckBoxCheck \(page 1033\)](#)
- [setCheckBoxHandler \(page 1035\)](#)

getChildren

Description

Summary

Returns an array of all the children of the specified parent.

Details

The getChildren command is useful in script procedures for automatically identifying the children of a node or nodes. The results are saved to a module array, which may in turn be passed to a subsequent command or procedure.

Functional area

Data retrieval

Command syntax

Syntax

```
getChildren parentName[-nodesOnly] [-fullPath] [-nameOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
nodesOnly	0	—	—	—
fullPath	0	—	nameOnly	—
nameOnly	0	—	fullPath	—

Return value

string array

Examples

```
string $mods[] = `getChildren rfoot`;
int $i;
for( $i = 0;
    $i < $mods.getCount();
    $i += 1 )
    print( $mods[$i] );
print( $mods.getCount());
// In this example, a module array called "$mods" is declared
// and the children of the node "right_foot" are assigned to it.
// The for loop iterates through the elements of the array and
// prints each one to the interface. The last line prints the
// value of $mods.getCount, which is an integer describing
// the number of children that were found in this instance.
```

Additional information

Related commands

- [getModules \(page 602\)](#)
- [getProperty \(page 655\)](#)

getClipboardText

Description

Summary

Get text from the clipboard

Details

Gets text that was copied to the clipboard.

Functional area

System

Command syntax

Syntax

```
getClipboardText
```

Arguments

None

Flags

None

Return value

string

Any text that was copied onto the system clipboard.

Examples

```
// Copy some text to the clipboard and then get it, verifying
// that they are the same
string $str;
copyString "Hello Shogun User";
$str = `getClipboardText`;

// Will print "Hello Shogun User"
print $str;
```

Additional information

Related commands

- [copyString \(page 228\)](#)

getClips

Description

Summary

Get a list of clips in the scene.

Details

Get a list of clips in the scene.

Functional area

Data retrieval

Command syntax

Syntax

```
getClips
```

Arguments

None

Flags

None

Return value

string array

Returns a string array of clip names

Examples

```
//create 2 Clips  
create Clip "Clip_A" "Clip_B";  
string $clips[] = `getClips`;  
print $clips;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

getColor

Description

Summary

Retrieves the draw color of various elements in Shogun Post.

Details

Retrieves the draw color of various elements in Shogun Post, as defined in the **Preferences**.

Use `default` to get the default color of a module when it is created. Modules draw in this color if they aren't selected and have data for the active clip.

Use `selected` to get the color that selected modules are drawn in.

Use `primarySelected` to get the color that the primary selection is drawn.

Use `markerLines` to get the default color of connection lines between markers, if one isn't specified.

Functional area

Data retrieval

Command syntax

Syntax

```
getColor selected | primarySelected | default | markerLines
```

Arguments

Name	Type	Required	Comments
<code>str</code>	string	yes	Can be one of the following values: <code>selected</code> , <code>primarySelected</code> , <code>default</code> , or <code>markerLines</code> . See <i>Description</i> above for more information about each value.

Flags

None

Return value

integer array

Examples

```
// Get the various element draw colors.
int $color[];

$color = `getColor "default"`;
print $color;
$color = `getColor "selected"`;
print $color;
$color = `getColor "primarySelected"`;
print $color;
$color = `getColor "markerLines"`;
print $color;
```

getColorPickerColor

Description

Summary

Get the user color picker color value.

Details

Gets the color value of the color picker user control specified by `userControlID`.

The return value is a three element integer array, holding the red, green, and blue components of the selected color. Each component ranges in value from 0 to 255.

Functional area

User Window

Command syntax

Syntax

```
getColorPickerColor userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

integer array

Examples

```
// Get the color value of a Color Picker User Control
int $windowId;
int $controlId;
int $color[];

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createColorPicker $windowId`;

// Get the default color
$color = `getColorPickerColor $controlId`;
print $color;
```

Additional information

Related commands

- [createColorPicker \(page 242\)](#)
- [setColorPickerColor \(page 1038\)](#)
- [setColorPickerHandler \(page 1040\)](#)

getConstraintError

Description

Summary

Get constraint error

Details

This method will return the error of the given constraint at the current time as determined by the difference between the constraining node's position and its offset. Value is in millimeters.

Functional area

Data retrieval

Command syntax

Syntax

```
getConstraintError "SolvingConstraint or LabelingConstraint name"
```

Arguments

Name	Type	Required	Comments
SolvingConstraint or LabelingConstraint name	string	yes	The constraint whose error is to be retrieved.

Flags

None

Return value

float

Returns the error of the given constraint at the current time.

Examples

```
// Will get the constraint error for the LFHD_Head Constraint.  
float $err;  
$err = `getConstraintError "LFHD_Head"`;  
print $err;
```

getConstraintOffset

Description

Summary

Gets the constraint offset for the specified node.

Details

The getConstraintOffset command gets the local constraint offset for the specified node. Used often during fine-tuning the setup of a skeleton to adjust the relationship between markers and bones.

Functional area

Data retrieval

Command syntax

Syntax

```
getConstraintOffset "constraintName" [-ws]
```

Arguments

Name	Type	Required	Comments
constraintName	string	yes	The name of the constraint whose offset is requested

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	0	—	—	Return the world space offset instead of the local space offset.

Return value

vector

Examples

```
// Get the world space position of the Constraint offset for the
// constraint LFHD_Head and print to the log.
vector $targetPos = `getConstraintOffset "LFHD_Head" -ws`;
print $targetPos;
```

Additional information

Related commands

- | [createSkelScript \(page 291\)](#)
- | [getConstraintPos \(page 481\)](#)
- | [setConstraint \(page 1043\)](#)
- | [snapToConstraint \(page 1209\)](#)
- | [solve \(page 1226\)](#)

getConstraintPos

Description

Summary

Get the constraint offset in world space.

Details

Get the world space value that represents where the constraints target would be if the solve had no error.

Functional area

Data retrieval

Command syntax

Syntax

```
getConstraintPos "SolvingConstraint or LabelingConstraint name"
```

Arguments

Name	Type	Required	Comments
SolvingConstraint or LabelingConstraint name	string	yes	Name of the constraint to operate on

Flags

None

Return value

vector

Returns a vector consisting of the world space value of the constraint offset.

Examples

```
// Get the world space position of the Constraint offset for the constraint  
LFHD_head  
vector $targetPos = getConstraintPos( "LFHD_head" );  
print $targetPos;
```

Additional information

Related commands

■ [getConstraintOffset \(page 479\)](#)

getConstraintsThisIsSource

Description

Summary

Gets the constraints that have the given module as a source.

Details

Returns a string array of names of constraints that have the given module as a source.

Functional area

Data retrieval

Command syntax

Syntax

```
getConstraintsThisIsSource "module" [-nameOnly] [-fullPath] [-solvingOnly]  
[-labelingOnly]
```

Arguments

Name	Type	Required	Comments
Module	string	yes	Module to retrieve list of constraints where module is a source.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
nameOnly	0	—	—	Only the name of each node is returned rather than the "minimum" path (the minimum needed to uniquely identify a node).
fullPath	0	—	—	The full path of each node is returned rather than the "minimum" path (the minimum needed to uniquely identify a node).
solvingOnly				
labelingOnly				

Return value

string array

Returns a string array of names of constraints that have the given module as a source.

Examples

```
// Get the constraints that have LFHD as a source.
string $constraints[] = `getConstraintsThisIsSource "LFHD"`;
print $constraints;
```

Additional information

Related commands

- | [attach \(page 150\)](#)
- | [getAttached \(page 453\)](#)
- | [getAttachedTo \(page 455\)](#)
- | [getConstraintsThisIsTarget \(page 485\)](#)
- | [getSolverBones \(page 687\)](#)
- | [setConstraint \(page 1043\)](#)

getConstraintsThisIsTarget

Description

Summary

Gets the constraints that have the given module as a target.

Details

Returns a string array of names of constraints that have the given module as a target

Functional area

Data retrieval

Command syntax

Syntax

```
getConstraintsThisIsTarget "module" [-fullPath] [-nameOnly]
```

Arguments

Name	Type	Required	Comments
Module	string	yes	Module to retrieve list of constraints where module is a target.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	—	Specifies that the full path of each node be returned rather than the "minimum" path, which is the minimum path needed to uniquely identify a node.
nameOnly	0	—	—	Specifies that only the name of each node be returned rather than the "minimum" path, which is the minimum path needed to uniquely identify a node.

Return value

string array

Returns a string array of names of constraints that have the given module as a target.

Examples

```
// Get the constraints that have the bone named Root as a target.
string $constraints[] = `getConstraintsThisIsTarget "Root"`;
print $constraints;
```

Additional information

Related commands

- | [attach \(page 150\)](#)
- | [getAttached \(page 453\)](#)
- | [getAttachedTo \(page 455\)](#)
- | [getConstraintsThisIsSource \(page 483\)](#)
- | [getSolverBones \(page 687\)](#)
- | [setConstraint \(page 1043\)](#)

getContributingCameras

Description

Summary

Gets a string array of contributing cameras for a selected marker.

Details

Enables you to automatically select cameras that contribute to a specified marker at the current frame.

Functional area

Data Retrieval

Command syntax

Syntax

```
getContributingCameras "marker" [-nameOnly] [-fullPath]
```

Arguments

Name	Type	Required	Comments
marker			

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
nameOnly	0	—	—	Specifies that only the name of the marker be returned.
fullPath	0	—	—	Specifies that the full path of the marker be returned rather than the "minimum" path, which is the minimum path needed to uniquely identify a marker.

Return value

String array

Additional information

Related commands

- [cameraView \(page 196\)](#)
- [removeRayContributions \(page 915\)](#)

getControlEnabled

Description

Summary

Returns the enabled status of a user control.

Details

Returns the enabled status of the user control specified by `userControlID`. A control that is disabled is blocked from user input (mouse and keystrokes). It is also dimmed to indicate a disabled status.

Functional area

User Window

Command syntax

Syntax

```
getControlEnabled userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to get the enabled status for.

Flags

None

Return value

boolean

Examples

```
// Get the enabled status of a User Control.
int $windowId;
int $controlId;
boolean $isEnabled;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Push Button Control in the window. We will disable it
// after we create it.
$controlId = `createPushButton $windowId -text "Disabled"`;

// Disable the control
enableControl $controlId false;

// Now get the enabled status. Should be false.
$isEnabled = `getControlEnabled $controlId`;
print $isEnabled;
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlText \(page 1051\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)
- | [showControl \(page 1195\)](#)

getControlPos

Description

Summary

Get the position of a user control.

Details

Gets the position of the user control specified by `userControlID`.

`intArray` holds the x,y coordinates of the 4 points of a rectangle, whose coordinates are relative to the top/left corner of the user window that the control is on.

Functional area

User Window

Command syntax

Syntax

```
getControlPos userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to get position for.

Flags

None

Return value

integer array

Examples

```
// Get the position and size of the User Control
int $windowId;
int $controlId;
int $rect[4];
string $height, $width;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Text Box Control in the window.
$controlId = `createTextBox $windowId`;

// Get the rect of the User Control. Since we haven't
// positioned it, it should be the default size, positioned
// at the top corner of the User Window (0, 0)
$rect = `getControlPos $controlId`;

// Calculate its width and height
$width = "width: " + string( $rect[2] - $rect[0] );
$height = "height: " + string( $rect[3] - $rect[1] );
print $width;
print $height;
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlText \(page 1051\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)
- | [showControl \(page 1195\)](#)

getControlText

Description

Summary

Get the text of a user control.

Details

Gets the text of the user control specified by `userControlID`.

Each user control has its own text which is usually displayed with the control. This text is not to be confused with tooltip text, which only gets displayed when a user hovers the mouse over the user control.

For text boxes, the control text is what the user edits. For other user control types, the text is for display only.

Some user controls, like color pickers, don't have any visible text, so this command doesn't have any effect on them.

All user controls have no text by default, unless created using the `-text` option.

Functional area

User Window

Command syntax

Syntax

```
getControlText userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to get text for.

Flags

None

Return value

string

Examples

```
// Set the text of a control.
int $windowId;
int $controlId;
string $controlText;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Text Box Control in the window.
$controlId = `createTextBox $windowId -text "Some Text"`;

// Get the text
$controlText = `getControlText $controlId`;
print $controlText;
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)
- | [showControl \(page 1195\)](#)

getControlVisibility

Description

Summary

Returns the visibility of a user control.

Details

Returns the visibility of the user control specified by `userControlID`.

All user controls are visible by default, unless created with the `-hidden` option.

Functional area

User Window

Command syntax

Syntax

```
getControlVisibility userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to get visibility for.

Flags

None

Return value

boolean

Examples

```
// Get the visibility of a control.
int $windowId;
int $controlId;
boolean $isShowing;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Push Button Control in the window. Create it hidden.
$controlId = `createPushButton $windowId -text "Sample" -hidden`;

// Get the visibility
$isShowing = `getControlVisibility $controlId`;
print $isShowing;
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlText \(page 1051\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)
- | [showControl \(page 1195\)](#)

getCount

Description

Summary

Returns the number of items in the given array. Returns 0 if the variable is not an array or has no elements.

Details

The getCount command helps you to manage different types of objects and arrays. Or, you can set up loops based on the returned number of objects of interest.

Functional area

Data retrieval

Command syntax

Syntax

```
getCount array
```

Arguments

Name	Type	Required	Comments
array_variable			Name of a module array

Flags

None

Return value

integer

Examples

```
string $bones[];  
$bones = `getModules -type "BoneNode"`;  
int $num = `getCount $bones`;  
print $num;  
// This script returns the number of BoneNodes in the scene.
```

Additional information

Related commands

■ [getModules \(page 602\)](#)

getCurrentChar

Description

Summary

Retrieves the name of the current character.

Details

Retrieves the name of the current character in the **Labeling** panel.

Functional area

Data retrieval

Command syntax

Syntax

```
getCurrentChar
```

Arguments

None

Flags

None

Return value

string

Returns a string with the name of the current character.

Examples

```
// Get the current Character.  
string $curCharacter = `getCurrentChar`;  
print $curCharacter;
```

Additional information

Related commands

- [getLabelerChars \(page 568\)](#)

getCurrentLabel

Description

Summary

Retrieves the name of the current label.

Details

Retrieves the name of the current label. The current label is what a marker will be labeled if a manual labeling operation is performed.

Functional area

Data retrieval

Command syntax

Syntax

```
getCurrentLabel
```

Arguments

None

Flags

None

Return value

string

Returns a string with the same of the current label.

Examples

```
// Get the current label.  
string $currentLabel = `getCurrentLabel`;  
print $currentLabel ;
```

getCurrentScript

Description

Summary

Get the name of the script currently in the Script Editor.

Details

Get the name of the script currently in the Script Editor, optionally providing the full path.

Functional area

Interface

Command syntax

Syntax

```
getCurrentScript [-fullPath]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	—	Gets the full path for the current script

Return value

string

Examples

```
//Print the full path of the current script to the log.  
string $s = `getCurrentScript -fullPath`;  
print $s;
```

Additional information

Related commands

- | [runScript \(page 941\)](#)
- | [setScriptPaths \(page 1151\)](#)

getDirList

Description

Summary

Get a list of the directories found in a named directory.

Details

Get a list of the directories found in a named directory.

You can optionally return only the subset which match a naming pattern using the `-pattern` flag. You can also search sub-directories of the directories using the `-recursive` flag.

Be sure to use forward slashes instead of back-slashes.

Functional area

System

Command syntax

Syntax

```
getDirList "searchDirectory" [-recursive] [-nameOnly] [-pattern string]
```

Arguments

Name	Type	Required	Comments
searchDirectory	string	yes	The folder for which to search for directories. Be sure to use forward slashes.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
recursive	0	—	—	Searches for all directories and sub-directories under the specified directory. By default sub-directories are not searched.
nameOnly	0	—	—	Lists only the names of the directories rather than their full paths (e.g. "Data" for a directory path of "C:/MyFiles/Data/")
pattern	1	string	—	Searches for folders which match a specific search criteria. By default all directories are returned. The pattern should use "wildcard" syntax (e.g. "Data*")

Return value

string array

Returns a string array with all of the directories found.

Examples

```
// Get a list of all of the directories found under my capture session.
// Also get the sub-directories.
string $dirList[] = `getDirList "C:/Data/MyProject001/" -recursive`;
print $dirList;
```

Additional information

Related commands

■ [getFileList \(page 528\)](#)

getDistance

Description

Summary

Measures the distance between two nodes in a scene.

Details

The default behavior (no flags used) returns the distance at the current frame. The `-avg`, `-min`, `-max`, `-range`, and `-sd` flags all return a value that's based on a number of frames. That number of frames is the play range by default unless the `-rangesOnly` option is used.

Functional area

Data retrieval

Command syntax

Syntax

```
getDistance node1 node2 [-avg] [-min] [-max] [-range] [-sd] [-rangesOnly]
[-mustHaveKeys]
```

Arguments

Name	Type	Required	Comments
node1	string	yes	Node to measure from
node2	string	yes	Node to measure to

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
-avg	0	—	-min, -max, -range, -sd	Returns the average distance
-min	0	—	-avg, -max, -range, -sd	Returns the minimum distance
-max	0	—	-avg, -min, -range, -sd	Returns the maximum distance
-range	0	—	-avg, -min, -max, -sd	Returns the difference of the min and max
-sd	0	—	-avg, -min, -max, -range	Returns the standard deviation
-rangesOnly	0	—	—	Computes distance calculations over the selected time ranges only
-mustHaveKeys	0	—	—	Only computes distance on frames where both nodes have a key present

Return value

float

Returns the distance in mm from node1 to node2

Examples

```
// create a clip
create Clip Clip_1;

// create two objects which we want to compute distances between
create Marker Marker_1;
create Bone Bone_1;

// move the bone 300 and 400 in x, y (respectively
setKey "Translation.X" 300 -onMod Bone_1;
setKey "Translation.Y" 400 -onMod Bone_1;

// show that the bone is 500 mm away from the marker as expected
print(getDistance("Marker_1", "Bone_1"));
```

Additional information

Related commands

■ [getLength \(page 576\)](#)

getDropListItem

Description

Summary

Get a user drop list item.

Details

Gets the string of an item in the drop list user control specified by `userControlID`. If an invalid index is specified, the command will fail.

Functional area

User Window

Command syntax

Syntax

```
getDropListItem userControlID itemIndex
```

Arguments

Name	Type	Required	Comments
<code>index</code>	int	yes	Index of the drop list item to get.
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

string

Examples

```
// Get the string of an item in the Drop List User Control
int $windowId;
int $controlId;
int $i, $numItems;
string $itemStr;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List.
addDropListItem $controlId "Item 1";
addDropListItem $controlId "Item 2";
addDropListItem $controlId "Item 3";

// Get the number of items.
$numItems = `getNumDropListItems $controlId`;

// Iterate through the list of items.
for( $i = 0;
    $i < $numItems;
    $i += 1 )
{
    $itemStr = `getDropListItem $controlId $i`;
    print $itemStr;
}
```

Additional information

Related commands

- [addDropListItem \(page 105\)](#)
- [createDropList \(page 247\)](#)
- [deleteAllDropListItems \(page 324\)](#)
- [deleteDropListItem \(page 336\)](#)
- [findDropListItem \(page 401\)](#)

-
- | [getDropListSelectedItem \(page 513\)](#)
 - | [getNumDropListItems \(page 612\)](#)
 - | [selectDropListItem \(page 982\)](#)
 - | [setDropListHandler \(page 1058\)](#)

getDropListSelItem

Description

Summary

Get the user drop list selected item.

Details

Gets the zero-based index of the selected item in the drop list user control specified by `userControlID`. If no item is selected, -1 is returned.

Functional area

User Window

Command syntax

Syntax

```
getDropListSelItem userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Get the selected Drop List User Control item
int $windowId;
int $controlId;
int $index;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List.
addDropListItem $controlId "Item 1";
addDropListItem $controlId "Item 2";
addDropListItem $controlId "Item 3";

// Select the second item
selectDropListItem $controlId 1;

// Get the selected item
$index = `getDropListSelItem $controlId`;
print $index;
```

Additional information

Related commands

- | [addDropListItem \(page 105\)](#)
- | [createDropList \(page 247\)](#)
- | [deleteAllDropListItems \(page 324\)](#)
- | [deleteDropListItem \(page 336\)](#)
- | [findDropListItem \(page 401\)](#)
- | [getDropListItem \(page 510\)](#)
- | [getNumDropListItems \(page 612\)](#)
- | [selectDropListItem \(page 982\)](#)
- | [setDropListHandler \(page 1058\)](#)

getEclipseActiveTrial

Description

Summary

Return the path of active trial from Eclipse database

Details

Return the full path of active trial from Eclipse database. If there is no selection or selection does not include any trial, the empty string is returned

Functional area

Data management

Command syntax

Syntax

```
getEclipseActiveTrial
```

Arguments

None

Flags

None

Return value

string

The returned string is a full path of active trial from Eclipse database

Examples

```
// Print active trial file from Eclipse database
string $trial = `getEclipseActiveTrial`;
print $trial;
```

Additional information

Related commands

- | [getEclipseAssociatedCalibration \(page 517\)](#)
- | [getEclipseAssociatedSubjects \(page 519\)](#)
- | [getEclipseMarkedTrials \(page 521\)](#)

getEclipseAssociatedCalibration

Description

Summary

Return the path of calibration file associated with active trial from Eclipse database

Details

Return the full path of calibration file associated with active trial from Eclipse database. If there is no selection or selection does not include any trial the empty string is returned

Functional area

Data management

Command syntax

Syntax

```
getEclipseAssociatedCalibration
```

Arguments

None

Flags

None

Return value

string

The returned string is a full path of calibration file associated with active trial from Eclipse database

Examples

```
// Print calibration file for active trial from Eclipse database
string $calibration = `getEclipseAssociatedCalibration`;
print $calibration;
```

Additional information

Related commands

- | [getEclipseActiveTrial \(page 515\)](#)
- | [getEclipseAssociatedSubjects \(page 519\)](#)
- | [getEclipseMarkedTrials \(page 521\)](#)

getEclipseAssociatedSubjects

Description

Summary

Return the array of subject and names from the active trial from Eclipse (data management) database.

Details

Return the array of subjects and names from the active trial from Eclipse database. If there is no selection, or the selection does not include any trial the empty array is returned.

Functional area

Data management

Command syntax

Syntax

```
getEclipseAssociatedSubjects
```

Arguments

None

Flags

None

Return value

string array

The array of string includes the subject; names of active trial from Eclipse database

Examples

```
// Print all subjects for active trial from Eclipse database
string $subjects[] = `getEclipseAssociatedSubjects`;
int $1 = `getCount $subjects`;
print ("Subjects count = " + string($1), $subjects);
```

Additional information

Related commands

- | [getEclipseActiveTrial \(page 515\)](#)
- | [getEclipseAssociatedCalibration \(page 517\)](#)
- | [getEclipseMarkedTrials \(page 521\)](#)

getEclipseMarkedTrials

Description

Summary

Return the array of full paths of all marked trials from Eclipse database

Details

Return the array of full paths of all marked trials from Eclipse database. If there are no marked trials the empty array is returned.

(To mark a trial, select the trial and press the SPACE key or use context menu.)

Functional area

Data management

Command syntax

Syntax

```
getEclipseMarkedTrials
```

Arguments

None

Flags

None

Return value

string array

The array of string includes the full paths of all marked trials from Eclipse database

Examples

```
// Print all marked trials from Eclipse database  
string $s[] = `getEclipseMarkedTrials`;  
print $s;
```

Additional information

Related commands

- [getEclipseActiveTrial \(page 515\)](#)
- [getEclipseAssociatedCalibration \(page 517\)](#)
- [getEclipseAssociatedSubjects \(page 519\)](#)

getEditTool

Description

Summary

Gets which editing tool is active.

Details

Gets the active editing tool type. Here is a list of editing tools and their type numbers:

Editing Tool	Type number
New Point	0
Linear	1
Bell	2
Flatten/Expand	3
Offset	4

Functional area

Interface

Command syntax

Syntax

```
getEditTool
```

Arguments

None

Flags

None

Return value

integer

Examples

```
// Get the current editing tool, and cycle to the next one.
int $tool, $next;

// Get the current one.
$tool = `getEditTool`;

// Figure out the next one.
$next = $tool + 1;
if( $next > 4 )
{
    $next = 0;
}
setEditTool $next;
```

Additional information

Related commands

- [graphView](#) (page 730)
- [manipulator](#) (page 789)
- [setEditTool](#) (page 1061)

getExtrapolateFingersAfterSolve

Description

Summary

Get extrapolate fingers after solve.

Functional area

Skeletal solving

Command syntax

Syntax

```
getExtrapolateFingersAfterSolve
```

Arguments

None

Flags

None

Return value

boolean

Additional information

Related commands

- | [extrapolateFingers \(page 371\)](#)
- | [setExtrapolateFingersAfterSolve \(page 1067\)](#)

getFileExtension

Description

Summary

Retrieves the extension portion of a file path, excluding the dot.

Details

Returns the extension portion of a file path. For example, for "c:/temp/take25.c3d", "c3d" will be returned.

Functional area

System

Command syntax

Syntax

```
getFileExtension "filePath"
```

Arguments

Name	Type	Required	Comments
filePath	string	yes	Full path to the file to get the extension for. Be sure to use forward slashes instead of back slashes.

Flags

None

Return value

string

Returns the file's extension.

Examples

```
// Get the extension of a file path.  
string $extension;  
string $path = "c:/temp/take25.c3d";  
$extension = `getFileExtension $path`;  
print $extension;
```

Additional information

Related commands

- | [getFileLocation \(page 531\)](#)
- | [getFileName \(page 533\)](#)
- | [getTitle \(page 541\)](#)

getFileList

Description

Summary

Retrieves a list of files in a directory.

Details

Returns the list of files in the specified directory. Can optionally search through all sub-directories, as well as only retrieve file names matching a certain pattern.

Functional area

System

Command syntax

Syntax

```
getFileList "searchDirectory"[-recursive] [-pattern string] [-nameOnly] [-  
sort string string]
```

Arguments

Name	Type	Required	Comments
searchDirectory	string	yes	The directory to search for files in.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>recursive</code>	0	—	—	Searches for files in sub-directories of the specified directory. By default sub-directories are not searched.
<code>pattern</code>	1	string	—	Specifies the file name pattern to match when searching for files. Default is to search for all files (".*")
<code>nameOnly</code>	0	—	—	Specifies that only the file name portion (including extension) be returned instead of the full file path (e.g. return "MyFile.hdf" for "C:/Data/MyFile.hdf").
<code>sort</code>	2	string	—	Specifies that the returned file list be sorted according to the given criteria. The first argument can be either "name", "path", "size", "created", "modified", and "accessed". The second argument can be either "asc" or "desc" for ascending and descending, respectively. By default the files are returned in the order in which they are encountered.

Return value

string array

Returns a string array with all of the files found.

Examples

```
// Get list of files in a directory.
int    $i, $count;
string $path = "c:/data/day/";
string $files[];

// Get all the c3d files in the directory
$files = `getFileList $path -pattern "*.c3d"`;
```

```
// Get the number of files returned
$count = `getCount $files`;

// Operate on each file
for( $i = 0; $i < $count;
    $i += 1 )
{
    // Do something with the file
    // ...
    // ...
}
```

Additional information

Related commands

- [getDirList \(page 505\)](#)
- [getFileExtension \(page 526\)](#)
- [getFileLocation \(page 531\)](#)
- [getFileName \(page 533\)](#)
- [getFileTitle \(page 541\)](#)

getFileLocation

Description

Summary

Retrieves the directory portion of a file path, including the ending forward slash.

Details

Returns the directory portion of a file path. For example, for "c:/temp/take25.c3d", "c:/temp/" will be returned.

Functional area

System

Command syntax

Syntax

```
getFileLocation "filePath"
```

Arguments

Name	Type	Required	Comments
filePath	string	yes	Full path to the file to get the location for. Be sure to use forward slashes instead of back-slashes.

Flags

None

Return value

string

Returns the directory portion of a file path, including the ending forward slash.

Examples

```
// Get the location of a file path.  
string $location;  
string $path = "c:/temp/take25.c3d";  
  
// Should be "c:/temp/"  
$location = `getFileLocation $path`;  
print $location;
```

Additional information

Related commands

- | [getFileExtension \(page 526\)](#)
- | [getFileName \(page 533\)](#)
- | [getFileTitle \(page 541\)](#)

getFileName

Description

Summary

Retrieves the name portion of a file path, including the extension.

Details

Returns the name portion of a file path. For example, for "c:/temp/take25.c3d", "take25.c3d" will be returned.

Functional area

System

Command syntax

Syntax

```
getFileName "filePath"
```

Arguments

Name	Type	Required	Comments
filePath	string	yes	Full path to the file to get the name for. Be sure to use forward slashes instead of back-slashes.

Flags

None

Return value

string

Returns the name portion of a file path, including the extension.

Examples

```
// Get the name of a file path.  
string $name;  
string $path = "c:/temp/take25.c3d";  
  
// Should print "take25.c3d"  
$name = `getFileName $path`;  
print $name;
```

Additional information

Related commands

- | [getFileExtension \(page 526\)](#)
- | [getFileLocation \(page 531\)](#)
- | [getFileTitle \(page 541\)](#)

getFilePath

Description

Summary

Returns the path of a file handle.

Details

Returns the path of a file pointed to by the given file handle. The file handle is one returned by the [fileOpen \(page 387\)](#) command.

Functional area

Disk I/O

Command syntax

Syntax

```
getFilePath fileID
```

Arguments

Name	Type	Required	Comments
fileID	integer	yes	The fileID is usually a variable created earlier in the script with the fileOpen command

Flags

None

Return value

string

Returns the full path to the file pointed to by the given file handle.

Examples

```
// Create a temp file query the file path and
// print it to the command log
// This creates the file c:/temp/temp.txt and returns the handle to us.
int $fileID = `fileOpen "c:/temp/temp.txt" "w"`;
string $filePath = `getFilepath $fileID`;

// Be sure to close the file.
fileClose $fileID;

// Now print out the file path
print $filePath;
```

Additional information

Related commands

- [fileClose \(page 385\)](#)
- [fileOpen \(page 387\)](#)

getFilePos

Description

Summary

Get the current file position

Details

Use this command to get how far you are into a file. The value is in bytes and you can use the file position value to "point" to other parts of the file. This command only makes sense in files opened in binary mode.

Functional area

Disk I/O

Command syntax

Syntax

```
getFilePos fileID
```

Arguments

Name	Type	Required	Comments
fileID	int	yes	The fileID is usually a variable created earlier in the script with the fileOpen (page 387) command

Flags

None

Return value

integer

Examples

```
int $fileID;
int $filePos;
string $str = "}";
$fileID = `fileOpen "C:/testfile.log" "w"`;

//set the file position to the 27th position...
setFilePos $fileID 27;

// get the file position and print it out to the command log and
// the 27th space in the text file.
$filePos = `getFilePos $fileID`;
writeInt $fileID $filePos;

//close the file.
fileClose $fileID;

// now if you open "C:/testfile.log" you should see "27" in the
// 27th space counting from 0.
```

Additional information

Related commands

■ [setFilePos \(page 1068\)](#)

getFiles

Description

Summary

Gets a list of the currently open files, as file IDs

Details

Use this command to get how a list of the files you have opened using the [fileOpen \(page 387\)](#) command. The list is returned as an array of integers, containing the file IDs of the open files.

Functional area

Disk I/O

Command syntax

Syntax

```
getFiles
```

Arguments

None

Flags

None

Return value

integer array

Examples

```
int $idArray[] = `getFiles`;  
int $count = `getCount $idArray`;  
  
// Print out the number of open files  
print ( "You have " + string( $count ) + " files open" );
```

Additional information

Related commands

- [fileClose \(page 385\)](#)
- [fileOpen \(page 387\)](#)

getFileTitle

Description

Summary

Retrieves the title portion of a file path, excluding the extension

Details

Returns the title portion of a file path. For example, for "c:/temp/take25.c3d", "take25" will be returned.

Functional area

System

Command syntax

Syntax

```
getFileTitle "filePath"
```

Arguments

Name	Type	Required	Comments
filePath	string	yes	Full path to the file to get the title for.

Flags

None

Return value

string

Returns the file title, which is the file name minus the extension.

Examples

```
// Get the title of a file path.  
string $title;  
string $path = "c:/temp/take25.c3d";  
$title = `getFileTitle $path`;  
print $title;
```

Additional information

Related commands

- [getFileExtension \(page 526\)](#)
- [getFileLocation \(page 531\)](#)
- [getFileName \(page 533\)](#)

getFloatProperty

Description

Summary

Retrieves a float value from an object.

Details

Because Shogun Post commands can only have one return type, there must be separate commands for retrieving different types of property data.

This command is used to retrieve channel or attributes values that are floating points (e.g. the Opacity attribute or TranslationX).

If `propertyName` represents a channel, the channel value at the current time is returned, unless the `-c` option is specified. It has the exact same functionality as the [getProperty \(page 655\)](#) command.

Functional area

Data retrieval

Command syntax

Syntax

```
getFloatProperty moduleName propertyName[-c] [-d]
```

Arguments

Name	Type	Required	Comments
<code>propertyName</code>	string	yes	Name of the property on the object.
<code>moduleName</code>	string	yes	Object to retrieve property value from.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
c	0	—	—	Retrieves the property's const value. Only has an effect if <code>propertyName</code> represents a channel (i.e. can be animated).
d	0	—	—	—

Return value

float

Examples

```
// Get some information about the "LFHD" marker.
float $opacity;
float $xTrans;
$opacity = `getFloatProperty "LFHD" "Opacity" `;
print $opacity;

$xTrans = `getFloatProperty "LFHD" "TranslationX" `;
print $xTrans;
```

Additional information

Related commands

- | [getBooleanArrayProperty \(page 459\)](#)
- | [getBooleanProperty \(page 461\)](#)
- | [getIntArrayProperty \(page 562\)](#)
- | [getIntProperty \(page 564\)](#)
- | [getProperty \(page 655\)](#)
- | [getStringProperty \(page 693\)](#)
- | [getVectorProperty \(page 717\)](#)
- | [setProperty \(page 1130\)](#)

getFocus

Description

Summary

Get the input focus, if it belongs to a user control.

Details

Gets the input focus, if it belongs to a user control. If no user control has the input focus, 0 is returned.

Only one control (either user or native) can have the input focus at any one time. The control with the input focus is the one that receives keyboard and mouse input.

You can set the focus to a user control so that it will be the recipient of keyboard and mouse messages. The input focus is lost when the user clicks outside of the control, or tabs outside of it.

Functional area

User Window

Command syntax

Syntax

```
getFocus
```

Arguments

None

Flags

None

Return value

integer

Examples

```
// Get the input focus.
int $windowId;
int $controlId;
int $focusId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Text Box Control in the window.
$controlId = `createTextBox $windowId`;

// Set the focus to our Text Box
setFocus $controlId;

// Get the focus. $focusId and $controlId should now match
$focusId = `getFocus`;

if( $focusId == $controlId )
{
    print "Match!";
}
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlTip \(page 1053\)](#)
- | [showControl \(page 1195\)](#)

getGaps

Description

Summary

Gets an array of all the gap ranges on a node.

Details

Retrieves an array of gap ranges on a node. Each gap takes up two elements in the array, for the start and end frame of the gap. Thus, the array size is twice the number of gaps on the node.

Functional area

Data retrieval

Command syntax

Syntax

```
getGaps node [-any] [-rot] [-sort] [-noFeedback]
```

Arguments

Name	Type	Required	Comments
nodeName	string	yes	Node to get gap ranges for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
any	0	—	—	Specifies that a gap can exist on any of the sub channels. The default is that the gap must exist on all sub channels.
rot	0	—	—	Specifies that the rotation channel be searched for gaps. The default is to search on the translation channel.
sort	0	—	—	Specifies that the gap ranges be sorted in ascending order of gap size. The default is to leave the gap ranges in chronological order.
noFeedback	0	—	—	—

Return value

integer array

Examples

```
// Get all the gaps on the RFWT.
int $gaps[];
int $i, $count;

// Get all the gaps.
$gaps = `getGaps "RFWT"`;

// Print out their sizes.
$count = `getCount $gaps`;
for( $i = 0;
    $i < $count;
    $i += 2 )
{
    print int( $gaps[$i+1] - $gaps[$i] + 1 );
}
```

Additional information

Related commands

- [fillGaps \(page 389\)](#)
- [findGap \(page 404\)](#)

getGlobalBooleanVar

Description

Summary

Retrieves the value of a specified boolean global variable.

Details

Shogun Post's scripting language supports global variables that are accessible across scripts within a session of Shogun Post. This command will retrieve the value of a specified global variable (identified by the name argument of the command) which stores a Boolean.

Functional area

Data retrieval

Command syntax

Syntax

```
getGlobalBooleanVar name
```

Arguments

Name	Type	Required	Comments
name	string	yes	Name of the variable to retrieve

Flags

None

Return value

boolean

Returns the Boolean value stored in the specified global variable. Can be true or false.

Examples

```
Script A
// Set the special process flag to be used by other scripts
setGlobalVar useSpecialProcess false;
...
Script B
// Check to make sure the special processing flag exists
if (`getGlobalVarExists useSpecialProcess`)
{
    // If it exists, use its value to determine what to do
    if (`getGlobalBooleanVar useSpecialProcess`)
    {
        print("Using special process");
    }
    else
    {
        print("Not using special process");
    }
}
else
{ print("Global variable useSpecialProcess not found");
}
```

Additional information

Related commands

- | [delGlobalVar \(page 356\)](#)
- | [getGlobalFloatVar \(page 552\)](#)
- | [getGlobalIntVar \(page 554\)](#)
- | [getGlobalStringVar \(page 556\)](#)
- | [getGlobalVarExists \(page 558\)](#)
- | [getGlobalVectorVar \(page 560\)](#)
- | [setGlobalVar \(page 1074\)](#)

getGlobalFloatVar

Description

Summary

Retrieves the value of a specified floating point global variable.

Details

Shogun's scripting language supports global variables that are accessible across scripts within a session of Shogun Post. This command will retrieve the value of a specified global variable (identified by the name argument of the command) which stores a double-precision float value.

Functional area

Data retrieval

Command syntax

Syntax

```
getGlobalFloatVar name
```

Arguments

Name	Type	Required	Comments
name	string	yes	Name of the variable to retrieve

Flags

None

Return value

float

Returns the (double precision) floating-point value stored in the specified global variable.

Examples

```
// Use the global myVar
float $localVar = 2.0;
print($localVar);
if (`getGlobalVarExists myVar`)
{
    // Override local variable with global floating point value
    $localVar = `getGlobalFloatVar myVar`;
}
print($localVar);
```

Additional information

Related commands

- [delGlobalVar \(page 356\)](#)
- [getGlobalBooleanVar \(page 550\)](#)
- [getGlobalIntVar \(page 554\)](#)
- [getGlobalStringVar \(page 556\)](#)
- [getGlobalVarExists \(page 558\)](#)
- [getGlobalVectorVar \(page 560\)](#)
- [setGlobalVar \(page 1074\)](#)

getGlobalIntVar

Description

Summary

Retrieves the value of a specified integer global variable.

Details

Shogun Post's scripting language supports global variables that are accessible across scripts within a session of Shogun Post. This command retrieves the value of a specified global variable (identified by the name argument of the command) which stores a signed 32-bit integer value.

The range of possible values is $-2,147,483,648$ to $2,147,483,647$.

Functional area

Data retrieval

Command syntax

Syntax

```
getGlobalIntVar name
```

Arguments

Name	Type	Required	Comments
name	string	yes	Name of the variable to retrieve

Flags

None

Return value

integer

Returns the integer value stored in the specified global variable.

Examples

```
// Use the global myVar
integer $localVar = 2;
print($localVar);
if (`getGlobalVarExists myVar`)
{
    // Override local variable with global integer value
    $localVar = `getGlobalFloatVar myVar`;
}
print($localVar);
```

Additional information

Related commands

- | [delGlobalVar \(page 356\)](#)
- | [getGlobalBooleanVar \(page 550\)](#)
- | [getGlobalStringVar \(page 556\)](#)
- | [getGlobalVarExists \(page 558\)](#)
- | [getGlobalVectorVar \(page 560\)](#)
- | [setGlobalVar \(page 1074\)](#)

getGlobalStringVar

Description

Summary

Retrieves the value of a specified string global variable.

Details

Shogun Post's scripting language supports global variables that are accessible across scripts within a session of Shogun Post. This command retrieves the value of a specified global variable (identified by the name argument of the command) which stores a string value.

Functional area

Data retrieval

Command syntax

Syntax

```
getGlobalStringVar name
```

Arguments

Name	Type	Required	Comments
name	string	yes	Name of the variable to retrieve

Flags

None

Return value

string

Examples

Script A

```
// Define a global variable  
setGlobalVar myVar "ShogunPost 1.0";
```

Script B

```
// Check to make sure global variable set in Script A still exists  
if (`getGlobalVarExists myVar`)  
{  
    // Use it  
    string $progName = `getGlobalStringVar myVar`;  
    print($progName);  
}
```

Additional information

Related commands

- | [delGlobalVar \(page 356\)](#)
- | [getGlobalBooleanVar \(page 550\)](#)
- | [getGlobalFloatVar \(page 552\)](#)
- | [getGlobalIntVar \(page 554\)](#)
- | [getGlobalVarExists \(page 558\)](#)
- | [getGlobalVectorVar \(page 560\)](#)
- | [setGlobalVar \(page 1074\)](#)

getGlobalVarExists

Description

Summary

Queries whether a specified global variable exists in the Shogun Post session or not.

Details

Shogun Post's scripting language supports global variables that are accessible across scripts within a session of Shogun Post. This command checks the current session of Shogun Post to determine if a global variable exists and has been set or not.

Functional area

System

Command syntax

Syntax

```
getGlobalVarExists name
```

Arguments

Name	Type	Required	Comments
name	string	yes	Name of the variable to query

Flags

None

Return value

boolean

Returns whether the global variable exists or not (true if the variable exists, false otherwise).

Examples

```
// Script A
// Define a global variable
setGlobalVar myVar "ShogunPost 1.0";

// Script B
// Check to make sure global variable set in Script A still exists
if( `getGlobalVarExists myVar` )
{
    // Use it
    string $progName = `getGlobalStringVar myVar`;
    print($progName);
}
```

Additional information

Related commands

- [delGlobalVar \(page 356\)](#)
- [getGlobalBooleanVar \(page 550\)](#)
- [getGlobalIntVar \(page 554\)](#)
- [getGlobalStringVar \(page 556\)](#)
- [getGlobalVectorVar \(page 560\)](#)
- [setGlobalVar \(page 1074\)](#)

getGlobalVectorVar

Description

Summary

Retrieves the value of a specified string global variable.

Details

Shogun Post's scripting language supports global variables that are accessible across scripts within a session of Shogun Post. This command retrieves the value of a specified global variable (identified by the name argument of the command) which stores a vector value. In Shogun Post, a vector is a fixed-size array of three double precision floating-point values used to store 3-dimensional data such as position and direction information.

Functional area

Data retrieval

Command syntax

Syntax

```
getGlobalVectorVar name
```

Arguments

Name	Type	Required	Comments
name	string	yes	Name of the variable to retrieve

Flags

None

Return value

vector

Returns the vector value stored in the specified global variable.

Examples

```
// Use the global myVar
vector $localVar = <<1.0, 2.0, 3.0>>;
print($localVar);
if (`getGlobalVarExists myVar`)
{
    // Override local variable with global vector value
    $localVar = `getGlobalFloatVar myVar`;
}
print($localVar);
```

Additional information

Related commands

- | [delGlobalVar \(page 356\)](#)
- | [getGlobalBooleanVar \(page 550\)](#)
- | [getGlobalFloatVar \(page 552\)](#)
- | [getGlobalIntVar \(page 554\)](#)
- | [getGlobalStringVar \(page 556\)](#)
- | [getGlobalVarExists \(page 558\)](#)
- | [setGlobalVar \(page 1074\)](#)

getIntArrayProperty

Description

Summary

Retrieves an array of integer values from an object's attributes.

Details

Because Shogun Post commands can only have one return type, there must be separate commands for retrieving different types of property data.

This command is mainly used to retrieve the color attribute values. Colors are returned as a three element integer array, holding the R, G, and B values of the color. Values range from 0-255.

Functional area

Data retrieval

Command syntax

Syntax

```
getIntArrayProperty moduleName propertyName
```

Arguments

Name	Type	Required	Comments
propertyName	string	yes	Name of the property on the object.
moduleName	string	yes	Object to retrieve property value from.

Flags

None

Return value

integer array

Examples

```
// Get an object's color.  
int $color[];  
$color = `getIntArrayProperty "LFHD" "Color"`;  
print $color;
```

Additional information

Related commands

- | [getBooleanArrayProperty \(page 459\)](#)
- | [getBooleanProperty \(page 461\)](#)
- | [getIntProperty \(page 564\)](#)
- | [getProperty \(page 655\)](#)
- | [getStringProperty \(page 693\)](#)
- | [getVectorProperty \(page 717\)](#)
- | [setProperty \(page 1130\)](#)

getIntProperty

Description

Summary

Retrieves an integer value from an object.

Details

Because Shogun Post commands can only have one return type, there must be separate commands for retrieving different types of property data.

This command is used to retrieve attributes values that are integers (e.g. `Frame_In` on characters) or time values (e.g. `Start_Frame` on clip).

It can also retrieve integer channel values. If `propertyName` represents a channel, the channel value at the current time is returned, unless the `-c` option is specified.

Functional area

Data retrieval

Command syntax

Syntax

```
getIntProperty moduleName propertyName[-c] [-d]
```

Arguments

Name	Type	Required	Comments
<code>propertyName</code>	string	yes	Name of the property on the object.
<code>moduleName</code>	string	yes	Object to retrieve property value from.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
c	0	—	—	Retrieves the property's const value. Only has an effect if propertyName represents a channel (i.e. can be animated).
d	0	—	—	—

Return value

integer

Examples

```
// Get some information about the "LFHD" marker
int $FrameIn = `getIntProperty "Actor_1" "Frame_In"`;
print $FrameIn;
```

Additional information

Related commands

- | [getBooleanArrayProperty \(page 459\)](#)
- | [getBooleanProperty \(page 461\)](#)
- | [getFloatProperty \(page 543\)](#)
- | [getIntArrayProperty \(page 562\)](#)
- | [getProperty \(page 655\)](#)
- | [getStringArrayProperty \(page 691\)](#)
- | [getStringProperty \(page 693\)](#)
- | [getVectorProperty \(page 717\)](#)
- | [setProperty \(page 1130\)](#)

getKeys

Description

Summary

Gets the frame numbers on which there are keys.

Details

Gets the frame numbers on which there are keys for the specified module and the specified channel.

Functional area

Data retrieval

Command syntax

Syntax

```
getKeys propertyName [-onMod string]
```

Arguments

Name	Type	Required	Comments
propertyName	string	yes	Name of the channel property to count keys for

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Returns the keys for the specified module only.

Return value

integer array

Examples

```
//print the frame numbers for all translation keys for the LFHD Marker  
int $keys[] = `getKeys Translation -onMod "LFHD"`;  
print $keys;
```

Additional information

Related commands

- [getNumKeys \(page 614\)](#)
- [getSelectedKeys \(page 672\)](#)
- [getTime \(page 701\)](#)

getLabelerChars

Description

Summary

Gets a list of characters in the labeler.

Details

Gets a list of the characters the labeler can label or unlabel.

Functional area

Data retrieval

Command syntax

Syntax

```
getLabelerChars
```

Arguments

None

Flags

None

Return value

string array

Returns a string array with the name of the labeler characters

Examples

```
// Get the labeler characters.  
string $labelerChars[] = `getLabelerChars`;  
print $labelerChars;
```

Additional information

Related commands

- [getCurrentChar \(page 499\)](#)

getLabelingClusterOffsets

Description

Summary

Returns vector containing offsets of given labeling clusters markers

Details

Functional area

Data retrieval

Command syntax

Syntax

```
getLabelingClusterOffsets "LabelingCluster name"
```

Return value

vector array

Additional information

Related commands

■ [setProperty \(page 1130\)](#)

getLastFile

Description

Summary

Return the name path of the file most recently loaded.

Details

Return the name of the last opened file. Useful when querying the scene for the last file opened. This gives you the last file imported if there was one imported into the scene.

Functional area

Data retrieval

Command syntax

Syntax

```
getLastFile [-noExtension]
```

Arguments

None

Flags

See above syntax.

Return value

string

Examples

```
//query the latest file path and  
//print it to the command log  
string $temp = `getLastFile`;  
print $temp ;
```

Additional information

Related commands

- | [resetClip \(page 931\)](#)
- | [setActiveClip \(page 1017\)](#)

getLastScript

Description

Summary

Gets last executed script.

Details

Functional area

System

Command syntax

Syntax

```
getLastScript
```

Arguments

None

Flags

None

Return value

string

getLayers

Description

Summary

Return the array of layer names of the active clip

Details

An error is generated if there is no clip in the scene.

Functional area

NLE

Command syntax

Syntax

```
getLayers
```

Arguments

None

Flags

None

Return value

string array

Examples

```
// Print the name of the active clip
string $clip = `getActiveClip`;
print $clip;

// Print the layers of the active clip
string $layers [];
$layers = `getLayers`;
print $layers;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

getLength

Description

Summary

Returns the length of a vector

Details

Use `getLength` when you need to obtain the length of a vector.

Given the position of an object in a variable of type vector, `getLength` returns the distance from that object to the origin.

Use [getDistance \(page 507\)](#) when you need to obtain the distance between two points specified as variables of type vector.

Functional area

Math

Command syntax

Syntax

```
getLength vector
```

Arguments

Name	Type	Required	Comments
vector	vector	yes	Vector variable

Flags

None

Return value

float

Returns a floating point value which is the length of the input vector v

Examples

```
// In this procedure, a vector variable is declared and getLength is
// used to return its length.
// The command returns a float value of: 5 (3-4-5 right triangle)
vector $v1 = <<3, 4, 0>>;
float $length = getLength($v1);
print( $length );
```

Additional information

Related commands

- [getDistance \(page 507\)](#)
- [setLength \(page 1086\)](#)

getListBoxItem

Description

Summary

Get a user list box item.

Details

Gets the string of an item in the list box user control specified by `userControlID`. If an invalid index is specified, the command will fail.

Functional area

User Window

Command syntax

Syntax

```
getListBoxItem userControlID itemIndex
```

Arguments

Name	Type	Required	Comments
<code>index</code>	int	yes	Index of the list box item to get.
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

string

Examples

```
// Get the string of an item in the List Box User Control
int $windowId;
int $controlId;
int $i, $numItems;
string $itemStr;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createListBox $windowId`;

// Add some items to the List Box.
addListBoxItem $controlId "Item 1";
addListBoxItem $controlId "Item 2";
addListBoxItem $controlId "Item 3";

// Get the number of items.
$numItems = `getNumListBoxItems $controlId`;

// Iterate through the list of items.
for( $i = 0;
    $i < $numItems;
    $i += 1 )
{
    $itemStr = `getListBoxItem $controlId $i`;
    print $itemStr;
}
```

Additional information

Related commands

- [addListBoxItem](#) (page 110)
- [createListBox](#) (page 259)
- [deleteAllListBoxItems](#) (page 326)
- [deleteListBoxItem](#) (page 340)
- [findListBoxItem](#) (page 407)

-
- [getListBoxSellItems \(page 581\)](#)
 - [getNumListBoxItems \(page 616\)](#)
 - [selectListBoxItem \(page 989\)](#)
 - [setListBoxHandler \(page 1088\)](#)

getListBoxSelItems

Description

Summary

Get the user list box selected items.

Details

Returns an array of zero based indices of the selected items of the list box user control specified by `userControlID`.

Because list box user controls can have multiple selected items (if the `-multi` flag was specified for the [createListBox](#) (page 259) command), an array must be returned.

If there are no selected items, an empty array is returned.

Functional area

User Window

Command syntax

Syntax

```
getListBoxSelItems userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

integer array

Examples

```
// Get the selected List Box User Control items
int $windowId;
int $controlId;
int $indices[];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createListBox $windowId -multi`;

// Add some items to the List Box.
addListBoxItem $controlId "Item 1";
addListBoxItem $controlId "Item 2";
addListBoxItem $controlId "Item 3";

// Select the second and third items
selectListBoxItem $controlId 1;
selectListBoxItem $controlId 2;

// Get the selected items
$indices = `getListBoxSelItems $controlId`;
print $indices;
```

Additional information

Related commands

- | [addListBoxItem \(page 110\)](#)
- | [createListBox \(page 259\)](#)
- | [deleteAllListBoxItems \(page 326\)](#)
- | [deleteListBoxItem \(page 340\)](#)
- | [findListBoxItem \(page 407\)](#)
- | [getListBoxItem \(page 578\)](#)
- | [getNumListBoxItems \(page 616\)](#)
- | [selectListBoxItem \(page 989\)](#)
- | [setListBoxHandler \(page 1088\)](#)

getListViewItemCheck

Description

Summary

Get the checked state of an item in a list view user control.

Details

Get the checked state of an item in a list view user control.

Functional area

User Window

Command syntax

Syntax

```
getListViewItemCheck userControlID itemRow
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the window to operate on
itemRow	integer	yes	Row number of the item in the list view

Flags

None

Return value

boolean

Returns the checked state of the item.

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkBoxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columns
string $columns[3];
int $widths[3];

$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the first
// column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
```



```
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

getListViewItemText

Description

Summary

Get List View item value

Details

This command will return the value of the text at a list view row and column location. Row and column index values are zero-based.

Functional area

User Window

Command syntax

Syntax

```
getListViewItemText userControlID itemRow itemColumn
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the user control to operate on
itemRow	integer	yes	Zero based row index of the item in the list view
itemColumn	integer	yes	Zero based column index of the item in the list view

Flags

None

Return value

string

Returns the string value of the list view item

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columns
string $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the first column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
```

```
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

getListViewSelItems

Description

Summary

Get the selected items in a list view user control

Details

This command returns an array of the indices of the selected items in a list view user control. The indices are zero-based. The indices are returned in the order that the items were selected in, meaning that they may not be in ascending order.

If the list view was created using the `-singleSel` flag, the command will return an array of size 1.

Functional area

User Window

Command syntax

Syntax

```
getListViewSelItems userControlID
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	The ID of the list view to operate on

Flags

None

Return value

integer array

Returns an array of the indices of the selected items in the list view

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;

setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columns
string $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the first
// column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
```

```
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewItemText \(page 586\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

getLogFile

Description

Summary

Get the path of the command logging file.

Details

The command logging file captures the output of the Shogun Post Log, storing all Shogun Post command feedback.

The command logging file is specified in the **Directories** section of the **Preferences** dialog or via the command [setLogFile \(page 1104\)](#).

Functional area

Data retrieval

Command syntax

Syntax

```
getLogFile
```

Arguments

None

Flags

None

Return value

string

Examples

```
//get the path of the command logging file into a variable  
string $lf = `getLogFile`;
```

```
//print the path to the log  
print $lf;
```

Additional information

Related commands

- [setLogFile \(page 1104\)](#)

getMarkerConnection

Description

Summary

Determines if there is a connecting line between two markers.

Details

Determines if there is a connecting line between two markers. Will return true if a connecting lines exists, false if one does not exist.

Functional area

Data retrieval

Command syntax

Syntax

```
getMarkerConnection "sourceMarker" "targetMarker"
```

Arguments

Name	Type	Required	Comments
sourceMarker	string	yes	Marker at one end of connecting line.
targetMarker	string	yes	Marker at other end of connecting line.

Flags

None

Return value

boolean

Returns true if there is a connecting line between the two markers or false if there is not.

Examples

```
// Determine if there is a connecting line
// between two markers.
boolean $hasLine;

// First set up some marker connections on the head
setMarkerConnection "LFHD" "RFHD";
setMarkerConnection "RFHD" "RBHD";

// Demonstrate the existence of the marker connection we just made.
$hasLine = `getMarkerConnection "LFHD" "RFHD"`;
print $hasLine; // Should be true

// Demonstrate the lack of a connection
$hasLine = `getMarkerConnection "LBHD" "RFHD"`;
print $hasLine; // Should be false
```

Additional information

Related commands

- [removeMarkerConnection \(page 909\)](#)
- [setMarkerConnection \(page 1106\)](#)

getMarkerConnectionColor

Description

Summary

Returns the color of a marker connection.

Details

Returns the color of the marker connection between the two given markers as an integer array consisting of RGB values.

Functional area

Data retrieval

Command syntax

Syntax

```
getMarkerConnectionColor "sourceMarker" "targetMarker"
```

Arguments

Name	Type	Required	Comments
sourceMarker	string	yes	Marker at one end of connecting line.
targetMarker	string	yes	Marker at other end of connecting line.

Flags

None

Return value

integer array

Returns an integer array of RGB color values for the connection line between two markers.

Examples

```
// Create a red connection line between LFHD and RFHD.
setMarkerConnection "LFHD" "RFHD" -color 255 0 0;

// Get the connection line color between LFHD and RFHD
// and print the RGB values.
int $color[] = `getMarkerConnectionColor "LFHD" "RFHD"`;
print $color;
```

Additional information

Related commands

■ [setMarkerConnection \(page 1106\)](#)

getModule

Description

Summary

Get the name and location of an object in a scene

Details

Functional area

Data retrieval

Command syntax

Syntax

```
getModule [-fullPath] [-nameOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	nameOnly	Returns full path to an object
nameOnly	0	—	fullPath	Returns only the name of an object

Return value

string

Examples

```
//Get selected module(s) to have data baked
string $sel[] = `getModules -selected`;
string $characterNode[] = `getModules -selected -type Character`;
```

getModuleRange

Description

Summary

Gets a modules frame range of data.

Details

Gets a module's range of data. This is the union of the frame ranges of all channel data. If the module has no data, 1 and -1 is returned.

Functional area

Data retrieval

Command syntax

Syntax

```
getModuleRange module or moduleArray[-i]
```

Arguments

Name	Type	Required	Comments
moduleName	string	yes	Module to get range for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
i	0	—	—	—

Return value

Integer array

Examples

```
// Get the RFWT's range of data.  
int $dataRange[];  
$dataRange = `getModuleRange "RFWT"`;  
print $dataRange;
```

getModules

Description

Summary

Returns an array of all modules currently in the scene

Details

The getModules command is useful in script procedures for automatically identifying the modules in the scene. The flags provide options for narrowing results based upon selection status, or module type.

The results of the command are saved to a module array which may in turn be passed to a subsequent command or procedure.

Functional area

Data retrieval

Command syntax

Syntax

```
getModules [-type string] [-primary] [-selected] [-fullPath] [-nameOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
type	1	string	—	Return only modules of the named type.

Name	Flag arguments	Argument type	Exclusive to	Comments
primary	0	—	selected	Return only the current primary selection.
selected	0	—	primary	Return only modules that are currently selected.
fullPath	0	—	nameOnly	—
nameOnly	0	—	fullPath	—

Return value

string array

Examples

```
string $mds[] = `getModules -type BoneNode`;
int $i;
for( $i = 0;
    $i < $mds.getCount();
    $i += 1 )
{
    print( $mds[$i] );
}
print ( $mds.getCount());

// In this small procedure, a module array called "$mds" is declared
// and all scene modules of the type BoneNode are assigned to it.
// The for loop iterates through the elements of the array and
// prints each one to the interface. The last line prints the value
// of $mds.getCount, an integer describing the number of modules
// that were found in this instance.
```

Additional information

Related commands

- [getChildren \(page 467\)](#)
- [getProperty \(page 655\)](#)

getModuleType

Description

Summary

Gets a module's type.

Details

Retrieves a module's type. When retrieving an array of modules from a command, the module types will need to be determined from a script.

Functional area

Data retrieval

Command syntax

Syntax

```
getModuleType moduleName
```

Arguments

Name	Type	Required	Comments
moduleName	string	yes	The module to get the type for.

Flags

None

Return value

string

Examples

```
// Get the selected modules in the scene and print out their types.
string $modules[];
string $type;
int $i, $count;

// First get the modules
$modules = `getModules -s`;

// Iterate through and print out their types
$count = `getCount $modules`;
for( $i = 0;
    $i < $count;
    $i += 1 )
{
    $type = `getModuleType $modules[ $i ]`;
    print $type;
}
```

Additional information

Related commands

- [getChildren \(page 467\)](#)
- [getModules \(page 602\)](#)
- [getParent \(page 639\)](#)

getNamespace

Description

Summary

Returns all namespaces to the left of the module name as a single string.

Details

All modules under the same parent can be considered in the namespace of that parent, using slashes (/) to separate each module's parent's name. getNamespace operates on all selected modules unless you use the `-onMod` flag to specify the module on which to operate by name.

Note that you may specify the path in commands that require a module name, but you may not specify the path as a module name in the `Name` attribute. You need to specify only what is required to uniquely identify the name.

Also note that getNamespace returns all namespaces to the left of the module name as a single string, for example, if run on the namespace Joe:Bill:LFHD then Joe:Bill is returned.

Functional area

Namespace

Command syntax

Syntax

```
getNamespace [-onMod string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies the module on which the command operates.

Return value

string

Example

```
// Print a namespace returned by the getNamespace command
string$namespace=`getNamespace`;
print$namespace;
```

Additional information

Related commands

- | [addNamespace \(page 117\)](#)
- | [getNamespaces \(page 608\)](#)
- | [removeNamespace \(page 911\)](#)
- | [replaceNamespace \(page 927\)](#)

getNamespaces

Description

Summary

Returns all namespaces to the left of the module name as a string array, with an element for each space name.

Details

All modules under the same parent can be considered in the namespace of that parent, using slashes (/) to separate each module's parent's name. `getNamespaces` operates on all selected modules unless you use the `-onMod` flag to specify the module on which to operate by name.

Note that you may specify the path in commands that require a module name, but you may not specify the path as a module name in the `Name` attribute.

You need to specify only what is required to uniquely identify the name.

Functional area

Namespace

Command syntax

Syntax

```
getNamespaces [-onMod string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies the module on which the command operates.

Return value

string array

Example

```
// Print a namespace array returned by the getNamespaces command
string$namespace[]=`getNamespaces`;
print$namespace;
```

Additional information

Related commands

- [addNamespace \(page 117\)](#)
- [getNamespace \(page 606\)](#)
- [removeNamespace \(page 911\)](#)
- [replaceNamespace \(page 927\)](#)

getNumBoxNum

Description

Summary

Get the user num box number value.

Details

Returns the number that appears in a num box user control specified by `userControlID`.

Num boxes are text boxes that restrict their input and output to numerical values. The number returned is a float, to accommodate both integer and floating point values. If [createNumBox \(page 266\)](#) was created with the `-flt` flag, then all values will be interpreted as floats.

Otherwise, values will be rounded down to the nearest integer (i.e. any decimal value will be truncated).

Functional area

User Window

Command syntax

Syntax

```
getNumBoxNum userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

float

Examples

```
// Get the value of a Num Box User Control
int $windowId;
int $controlId;
float $val;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createNumBox $windowId -flt -num -10.75`;

// Get the value of the Num Box
$val = `getNumBoxNum $controlId`;
print $val;
```

Additional information

Related commands

- [createNumBox \(page 266\)](#)
- [setNumBoxHandler \(page 1111\)](#)
- [setNumBoxNum \(page 1114\)](#)

getNumDropListItems

Description

Summary

Get the number of user drop list items.

Details

Gets the number of items in the drop list user control specified by `userControlID`.

Functional area

User Window

Command syntax

Syntax

```
getNumDropListItems userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Get the number of Drop List User Control items
int $windowId;
int $controlId;
int $numItems;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List.
addDropListItem $controlId "Item 1";
addDropListItem $controlId "Item 2";
addDropListItem $controlId "Item 3";

// Get the number of items
$numItems = `getNumDropListItems $controlId`;
print $numItems;
```

Additional information

Related commands

- | [addDropListItem \(page 105\)](#)
- | [createDropList \(page 247\)](#)
- | [deleteAllDropListItems \(page 324\)](#)
- | [deleteDropListItem \(page 336\)](#)
- | [findDropListItem \(page 401\)](#)
- | [getDropListItem \(page 510\)](#)
- | [getDropListSelectedItem \(page 513\)](#)
- | [selectDropListItem \(page 982\)](#)
- | [setDropListHandler \(page 1058\)](#)

getNumKeys

Description

Summary

Returns the number of frames on which a module has keys.

Details

Used to determine the number of frames on which the primary selected module has keys.

Functional area

Data retrieval

Command syntax

Syntax

```
getNumKeys propertyName[-s] [-onMod string]
```

Arguments

Name	Type	Required	Comments
propertyName	string	yes	Channel property

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
s	0	—	—	Selected keys only
onMod	1	string	—	Returns key count for the specified module.

Return value

integer

Examples

```
int $i;
select RKNE; $i = `getNumKeys "Translation"`;
print $i; //Output "4058"
selectRange 1 100;
selectKeys -ranges;
$i = `getNumKeys "Translation.X" -s`;
print $i; //Output "100"
select;
$i = `getNumKeys Translation -onMod RKNE`;
print $i; //Output "4058"
```

Additional information

Related commands

- [breakTangents \(page 177\)](#)
- [unbreakTangents \(page 1321\)](#)

getNumListBoxItems

Description

Summary

Get the number of user list box items.

Details

Gets the number of items in the list box user control specified by `userControlID`.

Functional area

User Window

Command syntax

Syntax

```
getNumListBoxItems userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Get the number of List Box User Control items
int $windowId;
int $controlId;
int $numItems;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createListBox $windowId`;

// Add some items to the List Box.
addListBoxItem $controlId "Item 1";
addListBoxItem $controlId "Item 2";
addListBoxItem $controlId "Item 3";

// Get the number of items
$numItems = `getNumListBoxItems $controlId`;
print $numItems;
```

Additional information

Related commands

- | [addListBoxItem \(page 110\)](#)
- | [createListBox \(page 259\)](#)
- | [deleteAllListBoxItems \(page 326\)](#)
- | [deleteListBoxItem \(page 340\)](#)
- | [findListBoxItem \(page 407\)](#)
- | [getListBoxItem \(page 578\)](#)
- | [getListBoxSellItems \(page 581\)](#)
- | [selectListBoxItem \(page 989\)](#)
- | [setListBoxHandler \(page 1088\)](#)

getNumListViewColumns

Description

Summary

Get the number of columns in a list view

Details

Use this command to get the number of columns in a list view user control.

Functional area

User Window

Command syntax

Syntax

```
getNumListViewColumns userControlID
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the list view to operate on

Flags

None

Return value

integer

Returns the number of columns in the list view

Examples

```
// Get the number of columns in a list view we created earlier
int $listViewID = `getGlobalIntVar "ListViewID"`;
print( `getNumListViewColumns $listViewID` );
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewItemText \(page 586\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

getNumListViewItems

Description

Summary

Get the number of items in a list view

Details

This command will return the total number of items in a list view.

Functional area

User Window

Command syntax

Syntax

```
getNumListViewItems userControlID
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	The ID of the list view to operate on

Flags

None

Return value

integer

Returns the total number of items in the list view

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columns
string $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the
// first column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";
```

```
// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewItemText \(page 586\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

getNumModules

Description

Summary

Returns the number of modules in the scene.

Details

Used to determine the number of modules that currently exist in Shogun Post. The command, without any flags, returns a count of every module in the scene.

Functional area

Data retrieval

Command syntax

Syntax

```
getNumModules [-type string] [-selected]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
type	1	string	—	Returns the number of modules of the specified type.
selected	0	—	—	Returns the number of currently selected modules.

Return value

integer

Examples

```
$i = `getNumModules`;
print $i; //Output "1243"
```

```
$i = `getNumModules -type Marker`;
print $i; //Output "828"
```

```
$i = `getNumModules -type Bone`;
print $i; //Output "123";
```

```
select RightUpLeg RKNE;
$i = `getNumModules -selected`;
print $i; //Output "2";
```

```
$i = `getNumModules -type Bone -selected`;
print $i; //Output "1";
```


getNumShelfTabs

Description

Summary

Returns an integer with the number of tabs on the button shelf

Details

Use getNumShelfTabs when you need to know exactly how many tabs are currently in the button shelf.

Functional area

Interface

Command syntax

Syntax

```
getNumShelfTabs
```

Arguments

None

Flags

None

Return value

integer

Examples

```
getNumShelfTabs;  
// Returns an integer value indicating the number of  
// tabs currently on the button shelf.
```

Additional information

Related commands

- | [createShelfTab \(page 289\)](#)
- | [deleteShelfTab \(page 350\)](#)

getParameter

Description

Summary

Returns the value of the given parameter.

Details

Parameters are stored on characters and used by bones and constraints. The `getParameter` command allows the value of a parameter to be accessed. By default, the command expects the character to be selected, but the `-onMod` option can be used to specify the character without having to select it.

Functional area

Parameters

Command syntax

Syntax

```
getParameter parameterName[-onMod string] [-solving]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies which character to look on for a parameter of the given name.
solving				

Return value

float

Returns the value of the parameter.

Examples

```
// Get the parameter UpperLegLength from the character Bob
float $upperLegLength = `getParameter UpperLegLength -onMod "Bob"`;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [getParameters \(page 633\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [selectByParameter \(page 968\)](#)
- | [setParameter \(page 1118\)](#)

getParameterExpression

Description

Summary

Gets the expression of the subject static parameter.

Details

The command returns a string that contains the expression used by the parameter. The string will be empty if no expression is in use. It is valid for static parameters only, as only static parameters can have expressions. If used on a dynamic parameter, the command will error stating this.

Functional area

Parameters

Command syntax

Syntax

```
getParameterExpression parameterName [-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	yes	The name of the static parameter whose expression you want to obtain

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	The character that owns the parameter should be selected. If it is not, the onMod flag may be used to specify it.
solving				By default, the command operates on parameters of the labeling setup, but the solving flag may be used to operate on the solving setup parameters.

Return value

string

getParameterPriorValue

Description

Summary

Gets the prior value of the subject parameter.

Details

This command takes one argument, which is a string, and must be the name of the parameter whose prior value you want to obtain.

Functional area

Parameters

Command syntax

Syntax

```
getParameterPriorValue parameterName [-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	yes	The name of the parameter whose prior value you want to obtain

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	The character that owns the parameter should be selected. If it is not, the onMod flag may be used to specify it.
solving				By default, the command operates on parameters of the labeling setup, but the solving flag may be used to operate on the solving setup parameters.

Return value

float

getParameters

Description

Summary

Returns a list of the names of the parameters of a character.

Details

Parameters are stored on characters and used by bones and constraints. The `getParameters` command returns a string array of the names of all of the parameter on a character.

By default, the command expects the character to be selected, but `-onMod` can be used to specify the character without having to select it.

Has options for static or dynamic parameters only.

Functional area

Parameters

Command syntax

Syntax

```
getParameters [-onMod string] [-staticOnly] [-dynamicOnly] [-solving]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies which character you want to get the parameters of.
staticOnly				
dynamicOnly				
solving				

Return value

string array

Returns a string array of parameter names.

Examples

```
// Get a list of all the parameters on the character named Bob.
string $params[] = `getParameters -onMod "Bob"`;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [getParameter \(page 627\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [selectByParameter \(page 968\)](#)
- | [setParameter \(page 1118\)](#)

getParameterType

Description

Summary

Gets the type of the subject parameter.

Details

This command takes one argument, which is a string, and must be the name of the parameter whose type you want to obtain. It returns a string that contains the type of the parameter: either static or dynamic.

Functional area

Parameters

Command syntax

Syntax

```
getParameterType parameterName[-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	yes	The name of the parameter whose type you want to obtain

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	The character that owns the parameter should be selected. If it is not, the onMod flag may be used to specify it.
solving				By default, the command operates on parameters of the labeling setup, but the solving flag may be used to operate on the solving setup parameters.

Return value

string

getParameterUserValue

Description

Summary

Gets the user value of the subject static parameter.

Details

This command takes one argument, which is a string, and must be the name of the parameter whose user value you want to obtain.

It is valid for static parameters only, as only static parameters can have user values. If used on a dynamic parameter, the command will produce an error stating this.

Functional area

Parameters

Command syntax

Syntax

```
getParameterUserValue parameterName [-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	yes	The name of the static parameter whose user value you want to obtain

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	The character that owns the parameter should be selected. If it is not, the onMod flag may be used to specify it.
solving				By default, the command operates on parameters of the labeling setup, but the solving flag may be used to operate on the solving setup parameters.

Return value

float

getParent

Description

Summary

Retrieves the parent of a module.

Details

Retrieves the parent of a module. If the module has no parent, an empty string, "", is returned.

Functional area

Data retrieval

Command syntax

Syntax

```
getParent module [-fullPath] [-nameOnly]
```

Arguments

Name	Type	Required	Comments
moduleName	string	yes	The module to get parent for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	nameOnly	—
nameOnly	0	—	fullPath	—

Return value

string

Examples

```
// Get the parent of the "rhumerus".  
string $parent;  
$parent = `getParent "rhumerus"`;  
print $parent;
```

Additional information

Related commands

- | [getChildren \(page 467\)](#)
- | [getModules \(page 602\)](#)
- | [parent \(page 826\)](#)
- | [unparent \(page 1331\)](#)

getPlayEnd

Description

Summary

Returns the play range end frame

Details

This command allows you to query your scene for the last frame of the play range.

Functional area

Data retrieval

Command syntax

Syntax

```
getPlayEnd
```

Arguments

None

Flags

None

Return value

integer

Returns the play ranges end time.

Examples

```
int $end = `getPlayEnd`;
print $end;
// Assigns the result of the getPlayEnd command to an integer
// variable called "$end"; prints the value of the variable
// to the interface.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

getPlayStart

Description

Summary

Returns the current play range start frame.

Details

This command allows you to query your scene for the first frame of the play range.

Functional area

Data retrieval

Command syntax

Syntax

```
getPlayStart
```

Arguments

None

Flags

None

Return value

integer

Returns the play ranges start frame;

Examples

```
int $strt = `getPlayStart`;
print $strt;
// Assigns the result of the getPlayStart command to an integer
// variable called "$strt".
// Prints the value of the variable to the interface.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

getPointClosestTo

Description

Summary

Identify the point on a vector closest to a line

Details

11 and 12 specify a line segment which goes through both points. This line segment is extended to be a line of infinite length. getPointClosestTo will orthogonally project *p* onto this line to return the point which is closest to *p* and 11-12.

Note that the point returned can be outside of the line segment 11-12.

Functional area

Math

Command syntax

Syntax

```
getPointClosestTo vector1 vector2 vector3
```

Arguments

Name	Type	Required	Comments
vector1	vector	yes	The first endpoint of the vector which is constructed from vector1 and vector2
vector2	vector	yes	The second endpoint of the vector which is constructed from vector1 and vector2
vector3	vector	yes	The point in space for which the user wants to find the closest point on the line from vector1 and vector2

Flags

None

Return value

vector

Returns a variable of type vector which is the point on the line from vector1 and vector2 that is closest to vector3.

Examples

```
vector $l1 = <<1, 0, 0>>;
vector $l2 = <<0, 1, 0>>;
vector $o = <<0, 0, 0>>;
// l1 and l2 form a line of slope -1 in the X-Y plane
// that goes through both points l1 and l2. o is the origin.
// The point on the line closest to the origin is <<0.5, 0.5, 0>>
print(getPointClosestTo($l1, $l2, $o));

// This example shows that you can get a point outside of the line
// segment delineated by l1-l2 (the closest point to the line segment
// would be l2, but the closest point to the line of infinite length
// is <<-0.5, 1.5, 0>>)
vector $p = <<-1, 1, 0>>;
print(getPointClosestTo($l1, $l2, $p));
```

Additional information

Related commands

■ [getAngleTo \(page 447\)](#)

getPosition

Description

Summary

Retrieves the local or world-space position of a node.

Details

Retrieves the position of a node at the current time.

Functional area

Data retrieval

Command syntax

Syntax

```
getPosition "moduleName" [-ws] [-inSpaceOf string] [-preTrans] [-default]
```

Arguments

Name	Type	Required	Comments
nodeName	string	yes	The node to get position for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	0	—	inSpaceOf	Specifies that the worldspace value should be retrieved. The default is to retrieve local.

Name	Flag arguments	Argument type	Exclusive to	Comments
inSpaceOf	1	string	ws	The string passed in with this flag should be a module in the scene. The position returned is relative to the module passed in with this flag from the module passed into the command.
preTrans	0	—	default	Gets the pre-translation of the module passed in. Pre-translation can be found in the Attributes panel.
default	0	—	preTrans	This will retrieve the default position for the module passed in. The default can be viewed in the Channels panel by using the toolbar button with a d.

Return value

vector

Examples

```
// Get the local and world space position
// of the rhumerus bone.
vector $localPos;
vector $wsPos;

// First the local
$localPos = `getPosition "rhumerus"`;
// And then the world space
$wsPos = `getPosition "rhumerus" -ws`;
print $localPos;
print $wsPos;
```

Additional information

Related commands

- [getRotation \(page 662\)](#)
- [getScale \(page 666\)](#)

getPriority

Description

Summary

Get the name of the priority node.

Details

When working with multiple characters using similar marker sets, the character with priority will be the character whose modules can be selected via script. `getPriority` will return the name of the character that is currently set to have priority.

Functional area

Data manipulators

Command syntax

Syntax

```
getPriority
```

Arguments

None

Flags

None

Return value

string

Examples

```
select Actor_1;  
setPriority;  
string $s = `getPriority`;  
print $s;  
//Output "Actor_1"
```

Additional information

Related commands

■ [setPriority \(page 1128\)](#)

getProfileInt

Description

Summary

Retrieves an integer value from the users profile.

Details

Reads an integer value from the users profile. Profile values are identified by their section (sections begin with [section name]) and by the entry (or key) name.

Section groupings allow values to be organized in logical groups. They also allow values with the same name to be differentiated between different groups.

Specifying `-file` allows the values to be read from a file of the users choice. The file does not have to have an "ini" extension.

Note that some users use the profile (or other file) as a "global variable" mechanism. While this is still supported, it is not recommended and has been superseded by actual global variables which allow you to have variables that persist between script executions.

For more information, see [setGlobalVar \(page 1074\)](#) (and related commands).

Functional area

System

Command syntax

Syntax

```
getProfileInt "sectionHeader" "entry" defaultValue[-file string]
```

Arguments

Name	Type	Required	Comments
sectionHeader	string	yes	Logical grouping of the value. This will find the line in the profile which looks like [sectionHeader]

Name	Type	Required	Comments
entry	string	yes	The entry name. This will be the first value on the line followed by an = sign.
defaultValue	integer	yes	Value to return if a value with the given section and name could not be found, or if there was an error accessing the profile.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Inputs from another text file. The default is to input from the user's profile (ini file).

Return value

integer

Examples

```
// Store some information in the profile and read it back in
int $value;

// Write 100 to the ini file
writeProfileInt "My Section" "KeyValue" 100;

// Now retrieve it. Return 0 if the value couldn't be found.
$value = `getProfileInt "My Section" "KeyValue" 0`;
print $value;
```

Additional information

Related commands

- [getProfileString \(page 653\)](#)
- [writeProfileInt \(page 1360\)](#)
- [writeProfileString \(page 1363\)](#)

getProfileString

Description

Summary

Retrieves a string value from the users profile.

Details

Reads a string value from the users profile. Profile values are identified by their section (sections begin with [section name]) and by the entry (or key) name.

Section groupings allow values to be organized in logical groups. They also allow values with the same name to be differentiated between different groups.

Specifying `-file` allows the values to be read from a file of the users choice. The file does not have to have an "ini" extension.

Note that some users use the profile (or other file) as a "global variable" mechanism. While this is still supported, it is not recommended and has been superseded by actual global variables which allow you to have variables that persist between script executions. See [setGlobalVar \(page 1074\)](#) (and related commands) for more information.

Functional area

System

Command syntax

Syntax

```
getProfileString "sectionHeader" "entry" "defaultValue"[-file string]
```

Arguments

Name	Type	Required	Comments
sectionHeader	string	yes	Logical grouping of the value. This will find the line in the profile which looks like [sectionHeader]

Name	Type	Required	Comments
entry	string	yes	The entry name. This will be the first value on the line followed by an = sign.
defaultValue	string	yes	Value to return if a value with the given section and name could not be found, or if there was an error accessing the profile.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Inputs from another text file. The default is to input from the user's profile (ini file).

Return value

string

Examples

```
// Store some information in the profile and read it back in
string $value;
// Write "Hello World" to the ini file
writeProfileString "My Section" "KeyValue" "Hello World";

// Now retrieve it. Return "ERROR" if the value couldn't be found.
$value = `getProfileString "My Section" "KeyValue" "ERROR"`;
print $value;
```

Additional information

Related commands

- [getProfileInt \(page 651\)](#)
- [writeProfileInt \(page 1360\)](#)
- [writeProfileString \(page 1363\)](#)

getProperty

Description

Summary

Returns the value of the named property on the named module at the current time.

Details

The getProperty command is useful in script procedures for automatically identifying the value of the property of a module. The called properties can be in either the **Attributes** or **Channels** panels.

The results may be saved to a variable, which may in turn be passed to a subsequent command or procedure.

Functional area

Data retrieval

Command syntax

Syntax

```
getProperty moduleName propertyName[-c] [-d]
```

Arguments

Name	Type	Required	Comments
moduleName propertyName			The arguments consist of a module name and any property name that applies for that module type.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
c	0	—	—	—
d	0	—	—	—

Return value

float

Examples

```
float $loc = `getProperty RBWT "TranslationX"`;
print $loc;
// Identifies the X position of the marker object RBWT and
// saves the XYZ values to a float variable called "$loc".
float $wt = `getProperty RUPA "Weight"`;
print $wt;
// Returns the weight of the RUPA to a floating
// point variable $wt.
```

Additional information

Related commands

- [getChildren \(page 467\)](#)
- [getModules \(page 602\)](#)

getRadioButtonCheck

Description

Summary

Get the user radio button check value.

Details

Gets the check value of the radio button user control specified by `userControlID`.

Radio buttons are essentially check boxes, but they operate in a group with other radio buttons, allowing for mutually exclusive choices.

Call this command on each radio button in the group to see which has the checked value (if any).

Functional area

User Window

Command syntax

Syntax

```
getRadioButtonCheck userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

boolean

Examples

```
// Get the checked value of a Radio Button User Control
int $windowId;
int $controlId1, $controlId2, $controlId3;
int $rect[ 4 ];

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Create the User Controls on the Window. Check the middle one.
$rect[ 0 ] = 20;
$rect[ 1 ] = 20;
$rect[ 2 ] = 100;
$rect[ 3 ] = 40;
$controlId1 = `createRadioButton $windowId -pos $rect -text "Radio 1"`;
$rect[ 1 ] = 50;
$rect[ 3 ] = 70;
$controlId2 = `createRadioButton $windowId -pos $rect -text "Radio 2" -check`;
$rect[ 1 ] = 80;
$rect[ 3 ] = 100;
$controlId3 = `createRadioButton $windowId -pos $rect -text "Radio 3"`;

// Find out which one is checked.
if( `getRadioButtonCheck $controlId1` == true )
{
    print "Radio 1 checked";
}
else if(`getRadioButtonCheck $controlId2` == true )
{
    print "Radio 2 checked";
}
else if(`getRadioButtonCheck $controlId3` == true )
{
    print "Radio 3 checked";
}
else
{
    print "None checked";
}
```

Additional information

Related commands

- [createRadioButton \(page 274\)](#)
- [setRadioButtonHandler \(page 1138\)](#)

getRate

Description

Summary

Retrieves the scene data rate, in frames per second.

Details

Retrieves the scene data rate, in frames per second.

Functional area

Data retrieval

Command syntax

Syntax

```
getRate
```

Arguments

None

Flags

None

Return value

float

Examples

```
// Set and get the scene rate
int $fps;
playOptions -fps 24;

//set play to 24fps
$fps = `getRate`;
print $fps;
```

Additional information

Related commands

- [playOptions \(page 834\)](#)

getRotation

Description

Summary

Retrieves the local or world-space rotation of a node.

Details

Retrieves the rotation of a node at the current time.

Functional area

Data retrieval

Command syntax

Syntax

```
getRotation "moduleName" [-ws] [-inSpaceOf string] [-preRot] [-def]
```

Arguments

Name	Type	Required	Comments
moduleName	string	yes	The module to get rotation for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	0	—	inSpaceOf	Specifies that the worldspace value should be retrieved. The default is to retrieve local.
inSpaceOf	1	string	ws	—
preRot	0	—	def	—
def	0	—	preRot	—

Return value

vector

Examples

```
// Get the local and world space rotation of the rhumerus bone.
vector $localRot;
vector $wsRot;

// First the local
$localRot = `getRotation "rhumerus"`;

// And then the world space
$wsRot = `getRotation "rhumerus" -ws`;
print $localRot;
print $wsRot;
```

Additional information

Related commands

- [getPosition \(page 647\)](#)
- [getScale \(page 666\)](#)

getSavePath

Description

Summary

Returns the default directory that Shogun Post's **Open** and **Save As** dialog boxes initialize to.

Details

Returns the default directory that Shogun Post's **Open** and **Save As** dialog boxes initialize to, or the default directory of the Export dialog if the `-e` flag is specified.

Shogun's **Open** and **Save** dialog boxes share their default values. However, the **Import** and **Export** dialog boxes have different default values. The values of these may be set using the [setSavePath \(page 1145\)](#) command. The values also get updated whenever you use the dialog boxes.

For all file and directory paths, use forward-slashes instead of back-slashes.

To get the full path of the currently loaded file, use `getSavePath` to get the location that the file was opened from, followed by [getSceneName \(page 668\)](#) to get the name.

Functional area

System

Command syntax

Syntax

```
getSavePath [-fullPath] [-e]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	—	This flag will add the scene name and ".vdf" extension to the returned path. By default, just the directory is returned.
e	0	—	—	This flag will return the default directory that Shogun Post's Export dialog initializes to instead of the Open and Save dialog boxes default.

Return value

string

Returns the full path to the default directory that Shogun Post's **Open** and **Save As** dialog boxes initialize to.

Examples

```
// Launch the file dialog and get the directory location that they chose
loadFile;

// Should return the directory of the file the user chose in the
// Open File dialog.
string $str = `getSavePath`;
print $str;
```

Additional information

Related commands

- [getSceneName \(page 668\)](#)
- [setSavePath \(page 1145\)](#)

getScale

Description

Summary

Retrieves the local or world-space scale of a node.

Details

Retrieves the scale of a node at the current time.

Functional area

Data retrieval

Command syntax

Syntax

```
getScale "moduleName" [-ws] [-inSpaceOf string] [-def]
```

Arguments

Name	Type	Required	Comments
nodeName	string	yes	The node to get scale for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	0	—	—	Specifies that the worldspace value should be retrieved. The default is to retrieve local.
inSpaceOf	1	string	ws	—
def	0	—	—	—

Return value

vector

Examples

```
// Get the local and world space scale of the rhumerus bone.
vector $localScale;
vector $wsScale;
// First the local
$localScale = `getScale "rhumerus"`;
// And then the world space
$wsScale = `getScale "rhumerus" -ws`;
print $localScale;
print $wsScale;
```

Additional information

Related commands

■ [getRotation \(page 662\)](#)

getSceneName

Description

Summary

Gets the current scene name.

Details

The scene name appears in the title bar of Shogun Post and it determines the default filename if the scene is saved. For a blank scene, the value of the scene name is an empty string. If a file is imported into the scene, the scene name is usually derived from that file name, excluding the file extension. Retrieving the scene name can be useful in custom scripts for saving scene files or exporting files.

Tip

To get the full path of the currently loaded file, use [getSavePath \(page 664\)](#) to get the location that the file was opened from, followed by `getSceneName` to get the name.

Functional area

System

Command syntax

Syntax

```
getSceneName
```

Arguments

None

Flags

None

Return value

String

Examples

```
//script to automatically set the scene name with an appendage
//for saving to a file
//get the current scene name
string $SceneName = `getSceneName`;

//create a string to append to the name
string $Append = "_copy";

//amend the scene name variable
$SceneName = $SceneName + $Append;

//check the scene name in the log
print $SceneName;

//set the scene name
setSceneName $SceneName;

//open the file save dialog with the new scene name for the file
saveFile;
```

Additional information

Related commands

- [getSavePath \(page 664\)](#)
- [setSceneName \(page 1149\)](#)

getScriptPaths

Description

Summary

Get the list of directories searched by Shogun Post to generate the internal list of scripts and pipelines.

Details

This list can also be viewed at the top of the **Directories** tab in the **Preferences** dialog.

Functional area

System

Command syntax

Syntax

```
getScriptPaths
```

Arguments

None

Flags

None

Return value

String array

Examples

```
string $paths[] = `getScriptPaths`;
```

Additional information

Related commands

- [addScriptPath \(page 123\)](#)
- [clearScriptPaths \(page 208\)](#)
- [deleteScriptPath \(page 346\)](#)

getSelectedKeys

Description

Summary

Gets the frame numbers on which there are selected keys.

Details

Gets the frame numbers on which there are selected keys for the selected or specified module and the specified channel.

Functional area

Data retrieval

Command syntax

Syntax

```
getSelectedKeys propertyName[-onMod string]
```

Arguments

Name	Type	Required	Comments
propertyName	string	yes	Name of the channel property to count keys for

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Returns the selected keys for the specified module.

Return value

Integer array

Examples

```
int $i[];
select RKNE;
selectRange 1 2;
selectRange 5 6 -a;
selectKeys -ranges;

$i= `getSelectedKeys Translation`;
print $i; //Output "1 2 5 6"

select;
$i= `getSelectedKeys Translation -onMod RKNE`;
print $i; //Output "1 2 5 6"
```

Additional information

Related commands

■ [selectKeys \(page 987\)](#)

getSelectedTimeRanges

Description

Summary

Returns an array of integer pairs indicating the current group of selected time ranges.

Details

Use `getSelectedTimeRanges` when you need to know the details regarding the current in/out time selections in your scene.

Functional area

Data retrieval

Command syntax

Syntax

```
getSelectedTimeRanges [-noFeedback]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
noFeedback	0	—	—	—

Return value

integer array

Returns an array of integer pairs indicating the current group of selected time ranges.

Examples

```
int $fSTR[];
$fSTR = `getSelectedTimeRanges`;
int $i;
int $j = 1;
print "Ranges:";
for( $i = 0;
    $i < (getCount($fSTR));
    $i += 2 )
{
    print $j;
    $j += 1;
    print( "in frame " + string ($fSTR [$i] ));
    print( "out frame " + string($fSTR [$i + 1]));
    print " ";
}
// Print a list representing the set of currently selected time ranges.
// The list will consist of integers, two elements for each selected
// time range found.
```

Additional information

Related commands

- [findBadData \(page 395\)](#)
- [findSelectedKeys \(page 414\)](#)

getSelectionFilter

Description

Summary

Checks the current value of the selection filter object.

Details

One of the active views must be a 3D view for this command to work. Returns a Boolean value (on/off)

Functional area

Interface

Command syntax

Syntax

```
getSelectionFilter "objectName"
```

Arguments

Name	Type	Required	Comments
objectName	string	yes	Name of the object or module whose selection state is to be retrieved.

Flags

None

Return value

boolean

Example

```
// Checking Marker selection filter state.  
Boolean $Value = `getSelectionFilter "Marker"`;
```

Additional information

Related commands

- | [getViewFilter \(page 719\)](#)
- | [setSelectionFilter \(page 1153\)](#)
- | [setViewFilter \(page 1187\)](#)

getSelectionSetNodes

Description

Summary

Gets the nodes in the given selection set.

Details

Selection sets are used to select a group of nodes that may or may not be in the scene by name. For example, a selection set called `setWaist` might contain all the names for the waist markers you might expect to encounter.

The `getSelectionSetNodes` command returns a list of all the nodes in the given selection set as a string array.

Functional area

Selection

Command syntax

Syntax

```
getSelectionSetNodes setName
```

Arguments

Name	Type	Required	Comments
setName	string	yes	Specifies the name of the selection set to get the nodes of.

Flags

None

Return value

string array

Returns a string array of nodes in the given selection set.

Examples

```
// Get all the nodes in the selection set setWaist.  
string $waistNodes = `getSelectionSetNodes "setWaist"`;
```

Additional information

Related commands

- | [getSelectionSets \(page 680\)](#)
- | [getSelectionSetSets \(page 682\)](#)
- | [isSelectionSet \(page 757\)](#)
- | [selectionSet \(page 984\)](#)
- | [selectSet \(page 1003\)](#)

getSelectionSets

Description

Summary

Get a list of the selection sets loaded in Shogun Post.

Details

Selection sets are used to select a group of nodes that may or may not be in the scene by name. For example , a selection set called `setWaist` might contain all the names for the waist markers you might expect to encounter.

The `getSelectionSets` command returns a list of all the selection sets currently loaded in Shogun Post.

Functional area

Selection

Command syntax

Syntax

```
getSelectionSets
```

Arguments

None

Flags

None

Return value

string array

Returns a string array of the names of the selection sets.

Examples

```
// Get a list of all the selection sets in Shogun.  
string $selectionSets[] = `getSelectionSets`;
```

Additional information

Related commands

- | [getSelectionSetNodes \(page 678\)](#)
- | [getSelectionSetSets \(page 682\)](#)
- | [isSelectionSet \(page 757\)](#)
- | [selectionSet \(page 984\)](#)
- | [selectSet \(page 1003\)](#)

getSelectionSetSets

Description

Summary

Gets a list of the sets used in the given selection set

Details

A selection set can contain other selection sets. The command `getSelectionSetSets` can be used to get the selection sets present in a given selection set. The sets present are returned as a string array.

Functional area

Selection

Command syntax

Syntax

```
getSelectionSetSets setName
```

Arguments

Name	Type	Required	Comments
setName	string	yes	The name of the selection set get to get a list of the selection sets it contains.

Flags

None

Return value

string array

Returns a string array of the names of all the selection sets that are part of the given selection set.

Examples

```
// Get a list of all the selection sets present in the setBody
// selection set.
string $setsInBodySet[] = `getSelectionSetSets "setBody"`;
```

Additional information

Related commands

- | [getSelectionSetNodes \(page 678\)](#)
- | [getSelectionSets \(page 680\)](#)
- | [isSelectionSet \(page 757\)](#)
- | [selectionSet \(page 984\)](#)
- | [selectSet \(page 1003\)](#)

getSession

Description

Summary

Returns the path of the active session as a string

Details

Functional area

Data management

Command syntax

Syntax

```
getSession
```

Arguments

None

Flags

None

Return value

string

getShelfTabs

Description

Summary

Get a list of the shelf tab names

Details

It is possible to create user button shelf tabs, in addition to the standard button shelf tabs. When any user button shelf tabs are present, the getShelfTabs command gets the names of the tabs into a string array.

Functional area

Interface

Command syntax

Syntax

```
getShelfTabs
```

Arguments

None

Flags

None

Return value

string array

Examples

```
//get the names of the user shelf tabs into an array  
string $st[] = `getShelfTabs`;
```

```
//print the names of tabs to the log  
print $st;
```

Additional information

Related commands

- [createShelfTab \(page 289\)](#)
- [deleteShelfTab \(page 350\)](#)

getSolverBones

Description

Summary

Retrieves a string array of the names of the bone nodes that a solver is acting on.

Details

For a specified solver, this script returns an array containing the names (including paths, if desired) of the bone nodes that the solver is associated with.

The default format for each bone node is to provide the minimum path (top level node and the name of the bone), but the full path or just the name only can be specified.

Functional area

Data retrieval

Command syntax

Syntax

```
getSolverBones solverName[-fullPath] [-nameOnly]
```

Arguments

Name	Type	Required	Comments
<code>solverName</code>	string	yes	Solver to retrieve list of BoneNodes for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
fullPath	0	—	nameOnly	Set this flag to fill each string in the array with the full path to the bone
nameOnly	0	—	fullPath	Set this flag to fill each string in the array with only the name of the bone

Return value

string array

A string array containing the names and/or paths of the bone nodes associated with the solver

Examples

```
// Get the BoneNodes a Solver the chest solver is acting on.
string $bones[];
$bones = `getSolverBones "chest_solver"`;
print $bones;
```

Additional information

Related commands

- | [attach \(page 150\)](#)
- | [setConstraint \(page 1043\)](#)
- | [solver \(page 1228\)](#)

getSolversFirst

Description

Summary

Returns the current state of the solvers after clip/blends.

Details

Returns the current state of the solvers after clip/blends. If this is true then solvers update before clips when the user issues a command.

This command is useful when you are blending and looping data.

Functional area

Skeletal solving

Command syntax

Syntax

```
getSolversFirst
```

Arguments

None

Flags

None

Return value

boolean

Examples

```
//Get and print the current state of the Solvers After Clip/Blends  
boolean $n = `getSolversFirst`;  
print $n;
```

Additional information

Related commands

■ [solve \(page 1226\)](#)

getStringArrayProperty

Description

Summary

Similar to [getStringProperty \(page 693\)](#), but for string array attribute types.

Details

Functional area

Data retrieval

Command syntax

Syntax

```
getStringArrayProperty moduleName propertyName
```

Arguments

Name	Type	Required	Comments
propertyName	string	yes	Name of the property on the object.
moduleName	string	yes	Object to retrieve property value from.

Flags

None

Return value

string array

Additional information

Related commands

- | [getBooleanArrayProperty \(page 459\)](#)
- | [getBooleanProperty \(page 461\)](#)
- | [getFloatProperty \(page 543\)](#)
- | [getIntArrayProperty \(page 562\)](#)
- | [getIntProperty \(page 564\)](#)
- | [getProperty \(page 655\)](#)
- | [getStringProperty \(page 693\)](#)
- | [getVectorProperty \(page 717\)](#)
- | [setProperty \(page 1130\)](#)

getStringProperty

Description

Summary

Retrieves a string value from an object's attributes.

Details

Because Shogun Post commands can only have one return type, there must be separate commands for retrieving different types of property data.

This command is used to retrieve attributes that can have be represented as strings (e.g. Rotation_Order, Other_Part, Start_Frame).

Functional area

Data retrieval

Command syntax

Syntax

```
getStringProperty moduleName propertyName[-fullPath]
```

Arguments

Name	Type	Required	Comments
propertyName	string	yes	Name of the property on the object.
moduleName	string	yes	Object to retrieve property value from.

Flags

See above syntax.

Return value

string

Examples

```
// Get some information about the "thorax" bone.
string $part;
string $rotOrder;
$part = `getStringProperty "thorax" "Other_Part"`;
print $part;
$rotOrder = `getStringProperty "thorax" "Rotation_Order"`;
print $rotOrder;
```

With a VSK and VSS present in the scene for an actor named GG:

```
// Print "LIEL" (the name of the marker which is pointed to in the
// Source attribute of LIEL_LeftArm).
string $str = `getStringProperty "GG Labeling LIEL_LeftArm" "Source"`;
print $str;

// Print "GG\LIEL" which is the full path to that same marker.
string $str = `getStringProperty "GG Labeling LIEL_LeftArm" "Source" -fullPath`;
print $str;
```

Additional information

Related commands

- | [getBooleanArrayProperty \(page 459\)](#)
- | [getBooleanProperty \(page 461\)](#)
- | [getFloatProperty \(page 543\)](#)
- | [getIntArrayProperty \(page 562\)](#)
- | [getIntProperty \(page 564\)](#)
- | [getProperty \(page 655\)](#)
- | [getVectorProperty \(page 717\)](#)
- | [setProperty \(page 1130\)](#)

getSystemTime

Description

Summary

Gets the system date and time, in seconds.

Details

Gets the system date and time in seconds, as an integer. Useful for determining execution time of a script or other length operations.

Passing the system time to [formatTime \(page 430\)](#) provides a textual representation of the time, useful for time stamps or other places where date and time are needed.

Functional area

System

Command syntax

Syntax

```
getSystemTime
```

Arguments

None

Flags

None

Return value

integer

Return the time as seconds elapsed since midnight, January 1, 1970

Examples

```
// Calculate how long it takes to go through a 1,000,000
// iteration loop. It's about 3 seconds on a 3ghz machine.
int $i, $startTime, $endTime;
string $timeStr;

// Get the start time
$startTime = `getSystemTime`;
$timeStr = `formatTime $startTime`;
print $timeStr;

// Do an empty, lengthy loop
while( $i < 1000000 )
{
    $i += 1;
}

// Get the end time
$endTime = `getSystemTime`;
$timeStr = `formatTime $endTime`;
print $timeStr;

// Now get the execution time
print int( $endTime - $startTime );
```

Additional information

Related commands

- [formatTime \(page 430\)](#)
- [system \(page 1303\)](#)

getTabItem

Description

Summary

Get the index of the specified tab.

Details

Get the index of the specified tab for further manipulation

Functional area

User Window

Command syntax

Syntax

```
getTabItem userControlID itemIndex
```

Arguments

Name	Type	Required	Comments
itemString	string	no	The name of the tab to be added
userControlId	int	yes	ID of user window to place the control on.

Flags

None

Return value

string

Examples

```
// Create a Tab and add an additional tab to the main tab.
int $windowId;
int $mainTab;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

//create a main tab controller then add a second and third tab
$mainTab = `createTab $windowId "Main" "Second" "Third"`;
int $secondTab = `addTab $mainTab "Fourth"`;

//get and print the third tab
string $ThirdTab = `getTabItem $mainTab 2`;
print $ThirdTab;
```

Additional information

Related commands

- | [addTab \(page 127\)](#)
- | [createTab \(page 300\)](#)
- | [deleteAllTabItems \(page 330\)](#)
- | [deleteTabItem \(page 352\)](#)
- | [getTabSelectedItem \(page 699\)](#)
- | [selectTabItem \(page 1007\)](#)
- | [setTabHandler \(page 1167\)](#)

getTabSelItem

Description

Summary

Get the index of the specified tab.

Details

Get the index of the specified tab for further manipulation

Functional area

User Window

Command syntax

Syntax

```
getTabSelItem userControlID
```

Arguments

Name	Type	Required	Comments
userControlId	int	yes	ID of the control to be queried.

Flags

None

Return value

integer

Examples

```
// Create a tab and add and additional tab to the main tab.
int $windowId;
int $mainTab;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

//create a main tab controller then add a second tab to the main tab
$mainTab = `createTab $windowId "Main" "Second" "Third" -sel 2`;
int $secondTab = `addTab $mainTab "Fourth"`;
int $activeTab = `getTabSelItem $mainTab`;
print $activeTab;
```

Additional information

Related commands

- | [addTab \(page 127\)](#)
- | [createTab \(page 300\)](#)
- | [deleteAllTabItems \(page 330\)](#)
- | [deleteTabItem \(page 352\)](#)
- | [getTabItem \(page 697\)](#)
- | [selectTabItem \(page 1007\)](#)
- | [setTabHandler \(page 1167\)](#)

getTime

Description

Summary

Returns the current frame number or time in seconds.

Details

The getTime command is used to identify the current time, in terms of frames or seconds.

Functional area

Data retrieval

Command syntax

Syntax

```
getTime [-s]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
s	0	—	—	Return the current time in seconds. Otherwise, time is returned as a frame number

Return value

float

Returns the current scene time in seconds or frames

Examples

```
int $time = `getTime`;
print $time;
// Identifies the current frame and saves the result to an
// integer variable.
```

```
float $time = `getTime`;
print $time;
// Identifies the current frame and saves the result to a
// floating point variable.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

getTimeBoxTime

Description

Summary

Get the user time box time value.

Details

Gets the time value of the time box user control specified by `userControlID`.

The return value is in frames, regardless of the time display style setting in the Preferences.

Functional area

User Window

Command syntax

Syntax

```
getTimeBoxTime userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Get the time value of a Time Box User Control
int $windowId;
int $controlId;
int $time;

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createTimeBox $windowId`;

// Set the time to some arbitrary frame
setTimeBoxTime $controlId 125;

// Get the time we just set
$time = `getTimeBoxTime $controlId`;
print $time;
```

Additional information

Related commands

- | [createTimeBox \(page 308\)](#)
- | [setTimeBoxHandler \(page 1175\)](#)
- | [setTimeBoxTime \(page 1178\)](#)

getTimecode

Description

Summary

Gets timecode.

Details

Gets timecode.

Functional area

Data Retrieval

Command syntax

Syntax

```
getTimecode
```

Arguments

None

Flags

None

Return value

string

getTimeCodeOffset

Description

Summary

Get a frame number for the timecode offset of current video file

Details

Get a frame number correspondent to SMPTE offset of current video file.

Return 0 if there is no video clip in video view.

Functional area

Data editing

Command syntax

Syntax

```
getTimeCodeOffset
```

Arguments

None

Flags

None

Return value

integer

Return a frame number corresponding to the timecode offset of the current video file.

Examples

```
// Print timecode offset for current video clip  
int $offset = `getTimeCodeOffset`;  
print $offset ;
```

getTimecodeStandard

Description

Summary

Gets the current timecode standard.

Details

The timecode standard determines the base frame rate to which Shogun Post can set capture or playback rates.

This command returns a string to describe the current timecode standard. Returned values can be ntsc, ntscDrop, pal, film, filmNtsc or other. The actual frame rate is a multiple of the base frame rate; this can be retrieved with the [getRate \(page 660\)](#) command.

Functional area

System

Command syntax

Syntax

```
getTimecodeStandard
```

Arguments

None

Flags

None

Return value

string

Examples

```
//get the current timecode standard
//and print to the log
string $TCStandard = `getTimecodeStandard`;
print $TCStandard;
```

```
//get the current frame rate
//and print to the log
float $FrameRate = `getRate`;
print $FrameRate;
```

Additional information

Related commands

- | [getRate \(page 660\)](#)
- | [setFrameRate \(page 1072\)](#)

getTopLevelForm

Description

Summary

Get the top-level form of the user window.

Details

Gets the top-level form of the user window specified by `windowID`. For a description of a form, see [createForm \(page 250\)](#).

Each user window has a form that can be used as the starting point for dynamically placing and sizing user controls. When the form ID is obtained, use [setControlAnchor \(page 1045\)](#) to attach the controls to the sides of the form rectangle. As the user window's size changes, the form changes size as well, positioning the user controls bound to it.

Forms can be nested, so forms created using [createForm \(page 250\)](#) can be anchored to the top-level form of a user window, to create a hierarchy of forms that dynamically manage user control layout.

Using forms to lay out user controls is easier and preferable to using [setControlPos \(page 1048\)](#), as the layout logic only needs to be applied at creation time, not each time the user window changes size.

Functional area

User Window

Command syntax

Syntax

```
getTopLevelForm windowID
```

Arguments

Name	Type	Required	Comments
windowID	int	yes	ID of user window to get the top-level form for.

Flags

None

Return value

integer

Examples

```
// Use the top-level Form to lay out User Controls.
int $windowId;
int $controlId;
int $formId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Get the top-level form ID
$formId = `getTopLevelForm $windowId`;

// Create a Text Box Control in the window. Add it to the
// top-level form we just got
$controlId = `createTextBox $windowId -form $formId`;

// Attach the text controls sides to the top level form,
// so that it stretches across the top of the Window
setControlAnchor $controlId "left" "left" 5;
setControlAnchor $controlId "right" "right" 5;
setControlAnchor $controlId "top" "top" 5;

// Issue the call to lay out the top level form. We only
// need to do this once
layoutForm $formId;
```

Additional information

Related commands

- [getControlPos \(page 491\)](#)
- [setControlPos \(page 1048\)](#)

getTrajectories

Description

Summary

Gets trajectory ranges of given marker.

Details

Given a marker, returns an array of integers consisting of pairs of frame numbers. Each pair marks the start and end frame of a trajectory. The total number of trajectories returned is half the array size.

Functional area

Data retrieval

Command syntax

Syntax

```
getTrajectories marker
```

Arguments

None

Flags

None

Return value

integer array

Examples

```
// Get the trajectory ranges for LFHD  
int $ranges[] = `getTrajectories "LFHD"`;  
print $ranges;
```

getUseAxiomSolving

Description

Summary

Gets the status of the `Axiom Solving` flag.

Details

The command returns true if the Axiom algorithm is to be used for solving, or false if the legacy solving algorithm will be used.

Functional area

Skeletal solving

Command syntax

Syntax

```
getUseAxiomSolving
```

Arguments

None

Flags

None

Return value

boolean

Examples

```
// Get the status of the Axiom Solving flag.  
string $d = `getUseAxiomSolving`;  
print $d;
```

Additional information

Related commands

- [setUseAxiomSolving \(page 1185\)](#)

getVectorProperty

Description

Summary

Retrieves a vector value from an object.

Details

Because Shogun Post commands can only have one return type, there must be separate commands for retrieving different types of property data.

This command is used to retrieve channel or attributes values that are vectors or rotations (e.g. the translation channel or Local_Axis_Offset).

If `propertyName` represents a channel, the channel value at the current time is returned, unless the `-c` option is specified.

Functional area

Data retrieval

Command syntax

Syntax

```
getVectorProperty moduleName propertyName[-c] [-d]
```

Arguments

Name	Type	Required	Comments
<code>propertyName</code>	string	yes	Name of the property on the object.
<code>moduleName</code>	string	yes	Object to retrieve property value from.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
c	0	—	—	Retrieves the property's const value. Only has an effect if <code>propertyName</code> represents a channel (i.e. can be animated).
d	0	—	—	—

Return value

vector

Examples

```
// Get some information about the "thorax" bone.
vector$constTrans;
vector$offsetRot;
//Get the Translation values of the thorax constants.
$constTrans = `getVectorProperty "thorax" "Translation" -c`;
print $constTrans;
// Get the local axis offset.
$offsetRot = `getVectorProperty "thorax" "Local_Axis_Offset"`;
print $offsetRot;
```

Additional information

Related commands

- | [getBooleanArrayProperty \(page 459\)](#)
- | [getBooleanProperty \(page 461\)](#)
- | [getFloatProperty \(page 543\)](#)
- | [getIntArrayProperty \(page 562\)](#)
- | [getIntProperty \(page 564\)](#)
- | [getProperty \(page 655\)](#)
- | [getStringProperty \(page 693\)](#)
- | [setProperty \(page 1130\)](#)

getViewFilter

Description

Checks the current value of the view filter object.

Functional area

Interface

Command syntax

Syntax

```
getViewFilter "Object Name";
```

Arguments

Name	Type	Required	Comments
Object Name			Name of view filter object

Flags

None

Return value

boolean

Additional information

Related commands

■ [setViewFilter \(page 1187\)](#)

getViewLayout

Description

Summary

Identify the view layout configuration currently in use.

Details

Returns integer 1–8 according to the top (1) to bottom (8) listing in the layout menus above the main viewport.

Functional area

Interface

Command syntax

Syntax

```
getViewLayout
```

Arguments

None

Flags

None

Return value

integer

getViewTypes

Description

Summary

Get the view types currently displayed.

Details

This command retrieves the current view types in use into a string array. The different types are perspective, graph, capture, data health, nle, video, front, back, left, right, top and bottom.

Functional area

Interface

Command syntax

Syntax

```
getViewTypes
```

Arguments

None

Flags

None

Return value

string array

Examples

```
//get the view types in use into an array  
string $vt[] = `getViewTypes`;
```

```
//print array to the log  
print $vt;
```

Additional information

Related commands

- [getViewLayout \(page 720\)](#)
- [viewLayout \(page 1342\)](#)

getWindowRect

Description

Summary

Get the rect of the user window.

Details

Gets the bounding rectangle of the User Window specified by `windowId`. The return value represents a rectangle holding the Left, Top, Right, and Bottom values of the user window. User controls placed with [setControlPos \(page 1048\)](#) or the `-pos` option on creation should be placed relative to the top left corner of the user window.

Functional area

User Window

Command syntax

Syntax

```
getWindowRect userWindowID
```

Arguments

Name	Type	Required	Comments
<code>userWindowID</code>	<code>int</code>	yes	ID of user window to get the bounding rectangle for.

Flags

None

Return value

integer array

Examples

```
// Get the bounding rect of a User Window, and place
// a control within it.
int $windowId;
int $controlId;
int $rect[4], $controlRect[4];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Text Box Control in the window.
$controlId = `createTextBox $windowId -text "Get Window Rect Sample"`;

// Get the rect of the User Window.
$rect = `getWindowRect $windowId`;

// Position the control in the middle of the User Window
$controlRect[0] = $rect[0];
$controlRect[1] = ($rect[1] + $rect[3]) / 2;
$controlRect[2] = $rect[2];
$controlRect[3] = $controlRect[1] + 24;
setControlPos $controlId $controlRect;
```

Additional information

Related commands

- [getControlPos \(page 491\)](#)
- [setControlPos \(page 1048\)](#)

getWindowVisibility

Description

Summary

Get visibility of a docking window.

Details

Gets the visibility of a docking window. The docking window can be either a Shogun Post panel or a user window.

Functional area

User Window

Command syntax

Syntax

```
getWindowVisibility "windowName"
```

Arguments

Name	Type	Required	Comments
windowName	string	yes	Name of the window to get visibility for. Can be one of a Shogun Post panel or a user window.

Flags

None

Return value

boolean

Examples

```
// Get the visibility of the Attributes panel
boolean $isVisible;
$isVisible = `getWindowVisibility "Attributes"`;
print $isVisible;
```

Additional information

Related commands

- [showWindow \(page 1197\)](#)

goToBasePose

Description

Summary

Sets the selected labeling or solving bones to their base pose.

Functional area

Skeletal solving

Command syntax

Syntax

```
goToBasePose
```

Arguments

None

Flags

None

Return value

void

goToPreferredPose

Description

Summary

Shows the preferred pose for all selected bones.

Details

The preferred pose is an attribute on a bone that tells the solver the most likely pose the bone will be in.

This command shows the preferred pose by setting a key at the current frame representing the preferred pose on all selected bones. Executing this command does not change the skeleton or its properties in any way; it's just a means of visualizing the preferred pose in the 3D view.

Functional area

Skeletal solving

Command syntax

Syntax

```
goToPreferredPose
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Show the preferred pose for all bones in the scene.  
selectByType BoneNode;  
showPreferredPose;
```

graphView

Description

Summary

Set graph view properties.

Details

The `graphView` command provides flexible modes for working with keys and curves in the graph view. All of the options listed for this command are also accessible from the **Graph** view.

Functional area

Interface

Command syntax

Syntax

```
graphView [-timeFollow boolean] [-valFollow boolean] [-single boolean] [-zoomValues integer] [-hideUnselected boolean] [-displayMode string] [-showAllClips boolean] [-selectKeys boolean] [-selectTime boolean] [-boxSelect boolean] [-x boolean] [-y boolean] [-z boolean] [-ws boolean] [-optimize boolean] [-pageTime boolean] [-selectOnPrimaryOnly boolean] [-select boolean] [-showGaps boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
timeFollow	1	boolean	—	Turn on/off time following; graph view focus follows current time
valFollow	1	boolean	—	Turn on/off value following
single	1	boolean	—	Graphs only the primary selected node items
zoomValues	1	integer	—	Point selection
hideUnselected	1	boolean	—	Show only selected keys on graph curves
displayMode	1	string	—	Switch graph display mode
showAllClips	1	boolean	—	—
selectKeys	1	boolean	—	Use mouse for key selection
selectTime	1	boolean	—	Use mouse for time selection (can be used concurrently with selectKeys)
boxSelect	1	boolean	—	Use box selection if on, otherwise defaults to in/out toggle
x	1	boolean	—	Show/hide X graph components
y	1	boolean	—	Show/hide Y graph components
z	1	boolean	—	Show/hide Z graph components

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	1	boolean	—	—
optimize	1	boolean	—	—
pageTime	1	boolean	—	—
selectOnPrimaryOnly	1	boolean	—	When selecting keys, will only select the primary selected module
select	1	boolean	—	—
showGaps		boolean		

Return value

void

Examples

```
graphView -zoomValues toggle;
// This command adjusts the zoom component to center
// the selected keyframe values in the Graph view.
```

Additional information

Related commands

- [cameraView \(page 196\)](#)
- [viewLayout \(page 1342\)](#)

grayscalefitOptions

Description

Summary

Set reconstructor options.

Details

Functional area

Data editing

Command syntax

Syntax

```
grayscalefitOptions [-grayscaleCircleFittingEnabled boolean]
[storeCircles boolean] [-fitMethod string] [fastfitQualityThreshold
float] [-slowfitQualityThreshold float] [fastfitSizeThreshold integer] [-
slowfitSizeThreshold integer] [slowfitIntensityThreshold float] [-
numThreads integer] [-reset] [fromRTK] [-toRTK]
```

Arguments

None

Flags

See above syntax.

Return value

void

hasKey

Description

Summary

Returns a Boolean indicating whether or not the specified node has a key at specified frame. If no frame is specified, the current frame is passed.

Details

The command `hasKey` returns a Boolean providing information on whether or not a specified node has a key at the requested frame. If you do not specify a specific frame, Shogun will use the current frame as a default. For example, this command will help if you are trying to determine whether or not there is a gap in the data of the marker you are examining.

Also, `hasKey` has the option of letting you determine whether or not an existing key is selected or not selected. For example, you may be trying to figure out not only if a given object has a key, but also whether or not that key has been selected by some other operation.

Functional area

Data retrieval

Command syntax

Syntax

```
hasKey "moduleName" "propertyName" [-selected] [-seconds integer] [-frame integer] [-allTime] [-any]
```

Arguments

Name	Type	Required	Comments
<code>propertyName</code>	string		Specifies the name of the property to query.
<code>moduleName</code>	string		Specifies the name of the module to query.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selected	0	—	allTime	Allows for a narrowing of the query. If this flag is used, a selected key on the indicated property of the indicated module must exist in order to return a TRUE value
seconds	1	integer	frame, allTime	Allows for the specification of a specific time in seconds to narrow the query
frame	1	integer	seconds, allTime	Allows for the specification of a specific key to narrow the query
all time	0	—	seconds, frame	Query all time
any	0	—	—	Query channels independently.
allTime	0	—	seconds, frame	—

Return value

boolean

Examples

```
boolean $ktrans = `hasKey "LFHD" "TranslationX" -frame 50`;
print $ktrans;
// returns a value of "True" if the module "LFHD" has a
// translation key at frame 50

boolean $krot = `hasKey "lfemur" "RotationX" -frame 20 -selected`;
print $krot;
// returns a value of "True" if the module "lfemur" has a
// selected rotation key at frame 20
```

Additional information

Related commands

- [exists \(page 367\)](#)
- [isSelected \(page 755\)](#)

hasParameter

Description

Summary

Determines if a parameter exists on a character.

Details

Returns true if the given parameter exists on the given character and false if it does not.

Functional area

Parameters

Command syntax

Syntax

```
hasParameter characterName parameterName [-solving]
```

Arguments

Name	Type	Required	Comments
characterName	String	Yes	Defines the name of the character to check if the parameter exists on.
parameterName	String	Yes	Defines the name of the parameter.

Flags

See above syntax.

Return value

boolean

Returns true if the character has the parameter and false if it doesn't.

Examples

```
// Check if the Character named "Bob" has a parameter named "UpperArmLength"
if(`hasParameter "Bob" "UpperArmLength"`)
{
    print "Bob has a parameter named UpperArmLength";
}
else
{
    print "Bob does not have a parameter named UpperArmLength";
}
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [setParameter \(page 1118\)](#)

help

Description

Summary

Launches Shogun help viewer.

Details

This command will either launch Shogun's online help or print the syntax for a command to the command log.

Functional area

System

Command syntax

Syntax

```
help ["topic"][-syntax]
```

Arguments

Name	Type	Required	Comments
topic	string	no	The command name or topic for which to show information. If you specify the <code>-syntax</code> flag, then this should be the name of a command. If you do not specify the <code>-syntax</code> flag, Shogun launches its online help pages and attempts to locate this topic.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
syntax	0	—	—	Prints the syntax for the topic to the command log. If you specify this flag you must specify a valid command name. Without this flag, Shogun launches its online help pages.

Return value

void

Examples

```
// Display the syntax for the "create" command in the Log line in
// the status bar and in the Log Viewer
help create -syntax;

// Open Shogun's Help documentation and look for an index item of
// the string "fill gaps"
help "fill gaps";
```

Additional information

Related commands

■ [listCommands \(page 773\)](#)

hide

Description

Summary

Turn off the visibility of modules in the scene.

Details

Use `hide` when you want to turn off the visibility of objects in your scene.

Using no arguments hides all modules. To hide just your current selection, use `hide -s`. Use `hide -u` to hide everything that is not selected, and `hide -primary` to hide only the primary selection.

Hidden objects will be completely invisible in the Perspective, Front, Right, and Top views; connectivity with other markers will be invisible as well. In the VPL view, modules will be dimmed.

Functional area

Data manipulators

Command syntax

Syntax

```
hide ["name1" "name2" ...] [-u] [-primary] [-t]
```

Arguments

Name	Type	Required	Comments
name1, name2		No	

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
s	0	—	u, primary	Hide selected only
u	0	—	s, primary	Hide unselected only
primary	0	—	s, u	Hide only primary selection
t	0	—	—	Toggle the state of visibility on all modules. doesn't respect other options
all	0	—	u, primary	—

Return value

void

Examples

```
hide -u;
// Hide all modules in the scene, except for the group
// that is currently selected. Hidden modules stay hidden
// until you employ the show command.
```

Additional information

Related commands

■ [show \(page 1193\)](#)

imageLabeler

Description

Summary

Set the parameters of the image for image labeler

Details

This command allows setting the path to the image, the image location on the screen, markers' color and other parameters.

The possible values for the image location are `screen`, `view` and `cursor`. The image will be located in the middle of the screen, in the middle of current view or near a cursor position accordingly.

For the cursor position the option `adjust` allows adjusting image position to fit on screen.

Functional area

Interface

Command syntax

Syntax

```
imageLabeler [-bitmap string] [-location string] [-adjust boolean] [-ensureVisible boolean] [-activeColor string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>bitmap</code>	1	string	—	Specifies the path to the image
<code>location</code>	1	string	—	Set the location of the image on the screen. The possible values are <code>screen</code> , <code>view</code> and <code>cursor</code> .
<code>adjust</code>	1	boolean	—	If <code>cursor</code> location is set, this option allows adjusting image position to fit on screen
<code>ensureVisible</code>	1	boolean	—	Allows to show/hide the Labeling panel
<code>activeColor</code>	1	string	—	Set the color of the markers

Return value

void

Examples

```
// Set a position of the image dialog
imageLabeler -location "cursor" -adjust true;

// Set image and color of the Markers
imageLabeler -bitmap "C:/Pictures/Actor.bmp" -activeColor "green";
```

Additional information

Related commands

- [autoLabel \(page 158\)](#)
- [autoLabelOptions \(page 161\)](#)
- [label \(page 763\)](#)
- [labelOptions \(page 765\)](#)

insertBone

Description

Details

Creates a new bone and parents it to the selected bone. The `reparentChild` option can be used to parent the child of the selected bone to the newly created bone.

Functional area

Skeletal solving

Command syntax

Syntax

```
insertBone [-reparentChild]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>reparentChild</code>			—	Parents the child of the selected bone to the newly created bone

Return value

void

isAxiomEnabled

Description

Summary

Returns the enabled status of Axiom.

Details

Returns the enabled status of Axiom (true if Axiom is enabled).

Functional area

Interface

Command syntax

Syntax

```
isAxiomEnabled
```

Arguments

None

Flags

None

Return value

boolean

isEndOfFile

Description

Summary

Details

Use this command to determine whether you are at the end of a file. This is useful when you are reading in a file and want to know if you have finished reading all the contents or gotten to the end.

readXXXX commands performed when at the end of a file will fail, causing your script to fail, so it is good practice to call this command often when reading a file.

Functional area

Disk I/O

Command syntax

Syntax

```
isEndOfFile fileID
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

None

Return value

boolean

Examples

```
int $fileID;  
// Obtain file ID earlier do some operation to the file  
// ...  
while( `isEndOfFile` == false )  
{  
    // Read in something  
}
```

Additional information

Related commands

■ [fileOpen \(page 387\)](#)

isFileBinary

Description

Summary

Gets whether a file was opened as a binary file.

Details

Use this command to determine if a file was opened as a binary file.

Functional area

Disk I/O

Command syntax

Syntax

```
isFileBinary fileID
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

None

Return value

boolean

Examples

```
int $fileID;
// Obtain file ID earlier do some operation to the file
// ...
// See if the file was opened in binary mode
if( `isFileBinary $fileID` == true )
    print "File is a binary file";
else
    print "File is a text file";
```

Additional information

Related commands

■ [fileOpen \(page 387\)](#)

isFileReadable

Description

Summary

Gets whether a file was opened in Read mode and can be read from.

Details

Use this command to determine if a file was opened in Read mode, i.e. for reading data from. Files that are opened in read mode cannot be written to, and vice-versa.

Functional area

Disk I/O

Command syntax

Syntax

```
isFileReadable fileID
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

None

Return value

boolean

Examples

```
int $fileID;
// Obtain file ID earlier do some operation to the file
// ...
// See if the file was opened in "Read Mode"
if( `isFileReadable $fileID` == true )
    print "Can read from the file";
else
    print "Cannot read from the file";
```

Additional information

Related commands

■ [fileOpen \(page 387\)](#)

isFileWriteable

Description

Summary

Gets whether a file was opened in Write mode and can be written to.

Details

Use this command to determine if a file was opened in Write mode, i.e. for writing data to. Files that are opened in write mode cannot be read from, and vice-versa.

Functional area

Disk I/O

Command syntax

Syntax

```
isFileWriteable fileID
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

None

Return value

boolean

Examples

```
int $fileID;
// Obtain file ID earlier do some operation to the file
// ...
// See if the file was opened in write mode
if( `isFileWriteable $fileID` == true )
    print "Can write to the file";
else
    print "Cannot write to the file";
```

Additional information

Related commands

■ [fileOpen \(page 387\)](#)

isSelected

Description

Summary

Returns Boolean according to whether the specified module is selected or not.

Details

The command `isSelected` returns a value of true or false depending on whether or not the indicated module is selected.

This command is only used with objects, not strings, as the command is only relevant if the object exists (if the object does not exist, it is not relevant to determine its status of being selected vs. not).

Functional area

Data retrieval

Command syntax

Syntax

```
isSelected module
```

Arguments

Name	Type	Required	Comments
module			Name of the module in the scene that you wish to query.

Flags

None

Return value

boolean

Examples

```
// Here are 3 versions of syntax that are equally valid
// and return the same information
boolean $x = Actor_1.isSelected();
boolean $y = isSelected( Actor_1 );
boolean $z = `isSelected Actor_1`;
print $x $y $z;
```

Additional information

Related commands

- | [exists](#) (page 367)
- | [hasKey](#) (page 734)

isSelectionSet

Description

Summary

Determines whether a named selection set exists in Shogun Post.

Details

The selection sets that are loaded into Shogun Post are determined by the selection set file specified in the **Preferences** and any selection sets you make via scripting.

The isSelectionSet command can be used to determine if a selection set of a given name exists in Shogun Post.

Functional area

Selection

Command syntax

Syntax

```
isSelectionSet setName
```

Arguments

Name	Type	Required	Comments
isSelectionSet	string	yes	The name of the selection set to look for.

Flags

None

Return value

boolean

Returns true if the given set exists, false if it does not.

Examples

```
// See if the selection set setLeftFoot exists in Shogun Post.
if(`isSelectionSet "setLeftFoot"` == true )
{
    print "The selection set setLeftFoot is present in Shogun Post";
}
```

Additional information

Related commands

- | [getSelectionSetNodes \(page 678\)](#)
- | [getSelectionSets \(page 680\)](#)
- | [getSelectionSetSets \(page 682\)](#)
- | [selectionSet \(page 984\)](#)
- | [selectSet \(page 1003\)](#)

kinematicLabelOptions

Description

Summary

Sets the options on the kinematic labeler.

Details

Sets the options on the kinematic labeler.

Two options are controlled by this command: the number of iterations that the kinematic labeler will run, and the rejection distances. The starting and ending rejection distances are used to generate the "cooling profile" of the simulated annealing optimization algorithm used by the kinematic labeler.

Functional area

Labeling

Command syntax

Syntax

```
kinematicLabelOptions [-numIterations integer] [-rejectionDistances float  
float] [-reset]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
numIterations	1	integer	—	Specifies the number of times the solver will iterate. The default is 9 iterations.
rejectionDistances	2	float	—	Specifies the starting and ending rejection distances. The defaults are 1000.0 and 40.0.
reset	0	—	—	Resets the options to default values.

Return value

void

Examples

```
// Set the number of iterations on the kinematic labeler higher.
kinematicLabelOptions -numIterations 12;
```

Additional information

Related commands

■ [kinLabel \(page 761\)](#)

kinLabel

Description

Summary

Kinematically labels markers constrained to the selected bone.

Details

This command uses the kinematic labeler to label the markers constrained to the selected bones at the current frame.

By comparing the kinematic model of the skeleton (bone lengths, degrees of freedom, and joint ranges) to the unlabeled marker data, the kinematic labeler attempts to fit the skeleton to the marker data using a simulated annealing optimization algorithm. The selected bones must both have the same root bone.

Functional area

Labeling

Command syntax

Syntax

```
kinLabel
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Kinematically label the marker constraints to the selected bones.  
kinLabel;
```

Additional information

Related commands

■ [kinematicLabelOptions \(page 759\)](#)

label

Description

Summary

Labels the selected trajectory

Details

Labels the selected trajectory by transferring its keys to a marker on the character currently being labeled. See [labelOptions \(page 765\)](#) for various ways the label can be applied.

If the `targetLabel` argument is not specified, the current label (in the Labeling panel) will be used.

Functional area

Data editing

Command syntax

Syntax

```
label ["targetLabel"]
```

Arguments

Name	Type	Required	Comments
<code>targetLabel</code>	string	no	The name to label the selected trajectory.

Flags

None

Return value

string

Returns the label of the selected marker.

Examples

```
// Label the selected marker as LFHD.  
label "LFHD";
```

Additional information

Related commands

■ [labelOptions \(page 765\)](#)

labelOptions

Description

Summary

Sets the labeling options.

Details

Sets labeling options that control how a label is applied to a trajectory, which character is being labeled, what label is being applied, etc.

This command is executed when options are changed in the **Label Options** section of the **Labeling** panel.

This command adds some functionality that is not in the **Labeling** panel and may be useful to hot key or add to a marking menu.

Functional area

Data editing

Command syntax

Syntax

```
labelOptions [-nextChar] [-prevChar] [-charParent] [-nextLabel] [-prevLabel] [-mode string] [-direction string] [-type string] [-advance boolean] [-curChar string] [-curLabel string] [-cliffTol float] [-cliffMaxGap integer] [-hide boolean] [-labelRadiusOffset float] [-unlabelRadiusOffset float] [-loadActors boolean] [-snapToSelection] [-nextUnlabeled] [-prevUnlabeled] [-autoVelocity boolean] [-advanceUnlabeled boolean] [-followSelection boolean] [-restoreDefaultSettings]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
nextChar	0	—	prevChar, charParent, curChar	Sets the character to be labeled to the next character.
prevChar	0	—	nextChar, charParent, curChar	Sets the character to be labeled to the previous character.
charParent	0	—	nextChar, prevChar, curChar	Sets the character parent of the first selected marker to be the character to be labeled.
nextLabel	0	—	prevLabel, curLabel, nextUnlabeled, prevUnlabeled	Sets the name the unlabeled marker is labeled as to the next label in the list.
prevLabel	0	—	nextLabel, curLabel, nextUnlabeled, prevUnlabeled	Sets the name the unlabeled marker is labeled as to the previous label in the list.
mode	1	string	—	Specifies how the current label advances through the labels list after applying a labeling.
direction	1	string	—	Specifies if a label made at the current frame is extended to the trajectory start and/or end frames.
type	1	string	—	Specifies how a label made at the current frame extends to the trajectory start and end. If the markers are

Name	Flag arguments	Argument type	Exclusive to	Comments
				packed, also specifies if the label applies to other trajectories on the marker being labeled.
advance	1	boolean	—	Specifies if the current label in the label list advances after performing a label.
curChar	1	string	nextChar, prevChar, charParent	Specifies the character to label by name.
curLabel	1	string	nextLabel, prevLabel	Specifies the current label to apply to the selected marker by name.
cliffTol	1	float	—	Specifies the allowable amount of acceleration between frames for a trajectory to be considered continuous when labeling using cliff mode.
cliffMaxGap	1	integer	—	Specifies the max number of frames without data between trajectories on the marker being labeled for the labeler to label both trajectories. Only applies when using cliff mode.
hide	1	boolean	—	Hides or shows the Labeling panel.
labelRadiusOffset	1	float	—	Point size offset applied to markers when labeled.
unlabelRadiusOffset	1	float	—	Point size offset applied to markers when unlabeled.

Name	Flag arguments	Argument type	Exclusive to	Comments
loadActors	1	boolean	—	Specifies if characters with names Actor_* are available for labeling.
snapToSelection	0	—	—	Sets the current character to the character parent of the selected marker and sets the current label to the name of the selected marker.
nextUnlabeled	0	—	nextLabel, prevLabel, curLabel, prevUnlabeled	Sets the current label to the next label that is not already labeled.
prevUnlabeled	0	—	nextLabel, prevLabel, curLabel, nextUnlabeled	Sets the current label to the previous label that is not already labeled.
autoVelocity	1	boolean	—	Specifies if a velocity label operation should be performed after a label is applied.
advanceUnlabeled	1	boolean	—	Specifies if the current label in the labels list should auto advance to the next unlabeled label after a label is applied.
followSelection	1	boolean	—	Specifies if the current label in the labels list should automatically match the current select as selection changes in the scene.

Return value

void

Examples

```
// Set up the labeler to label the character named Bob using the
// cliff setting.
labelOptions -curChar "Bob";
labelOptions -type "Cliff";
```

Additional information

Related commands

■ [label \(page 763\)](#)

layoutForm

Description

Summary

Dynamically positions and/or sizes the user controls that a form is managing.

Details

Dynamically positions and/or sizes the user controls managed by `controlID`. For a description of forms, see [createForm \(page 250\)](#).

The form dynamically lays out all user controls anchored to it, or managed by it, based on its current size and position.

For top-level forms, the form's size and position is the bounding rectangle of the user window.

For forms created using [createForm \(page 250\)](#), the size and position is either set explicitly using [setControlPos \(page 1048\)](#), or automatically, if the form is being managed and anchored to another form.

Top-level forms automatically lay themselves out when a user window is sized and/or docked /undocked, so using [layoutForm \(page 770\)](#) is unnecessary. Top-level forms only need to have this command called on them once, at the end of the creation script, after all the layout logic has been specified using [setControlAnchor \(page 1045\)](#). Non-top-level forms need to have `layoutForm` called for them if they are not being managed by another form.

Functional area

User Window

Command syntax

Syntax

```
layoutForm userControlID
```

Arguments

Name	Type	Required	Comments
controlID	int	yes	ID of form user control to lay out.

Flags

None

Return value

void

Examples

```
// Anchor two controls two each other, and to the form top-level
int $windowId;
int $staticId, $textId;
int $formId;

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Get the top-level form ID
$formId = `getTopLevelForm $windowId`;

// Create a label User Control, and add it to the form
$staticId = `createStaticBox $windowId -text "Name" -form $formId`;

// Create a Text Box Control in the window. Also add it to the
// top-level form.
$textId = `createTextBox $windowId -form $formId`;

// Attach the Static User Control to the top and to the left
// of the top-level form
setControlAnchor $staticId "left" "left" 0;
setControlAnchor $staticId "top" "top" 0;

// Make the Text Box User Control stretch from the right side
// of the Static Box, to the right side of the User Window
setControlAnchor $textId "top" "top" 0 -target $staticId;
setControlAnchor $textId "left" "right" 5 -target $staticId;
setControlAnchor $textId "right" "right" 0;
```

```
// Issue the call to lay out the top level form. We only  
// need to do this once  
layoutForm $formId;
```

Additional information

Related commands

- | [createForm \(page 250\)](#)
- | [getControlPos \(page 491\)](#)
- | [getTopLevelForm \(page 710\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)

listCommands

Description

Summary

Prints all the commands to the log window sorted by category.

Details

Use listCommands when you need to check for the presence of a command inside your version of Shogun Post. You can then use help to get more detailed information on a specific command.

Functional area

System

Command syntax

Syntax

```
listCommands [-category string] [-reverse] [-syntax] [-native] [-menu] [-  
commands] [-categories]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
category	1	string	commands	Displays the commands for the given category. Use -categories to see what are the valid names to use for this flag argument.
reverse	0	—	—	Prints the commands in reverse order
syntax	0	—	—	Also display the commands syntax
native	0	—	menu	Only display the native commands
menu	0	—	native	Only display menu or GUI commands
commands	0	—	category	Do not sort the commands by category
categories	0	—	—	Display the list of categories
xmlDump	1	string	—	Generates XML files for each native Shogun command

Return value

void

Examples

```
// Lists all commands, including syntax descriptions for each one.
listCommands -syntax;
```

Additional information

Related commands

■ [help \(page 739\)](#)

listObjectTypes

Description

Summary

Lists attribute and channel name information for all module types in Shogun Post.

Details

Lists all of the object, or module types available in Shogun Post. Also lists all of the channels and attributes that a module type has.

Functional area

System

Command syntax

Syntax

```
listObjectTypes [-filename string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
filename	1	string	—	Will redirect all output to the specified file. Default is to print to the Log. Use forward-slashes for all file paths.

Return value

void

Examples

```
// Generate a report of all of Shogun Post's module types and their  
// attributes and channels  
listObjectTypes -filename "C:/Output.txt";
```


listParameters

Description

Summary

Lists the parameters on a character.

Details

Prints the parameters on the selected character to the command log. If -mod is used the parameters on the character specified will be printed. Has options for static or dynamic parameters only.

Functional area

Parameters

Command syntax

Syntax

```
listParameters [-onMod string] [-staticOnly] [-dynamicOnly] [-solving]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Defines the name of the character whose parameters will be listed.
staticOnly				Prints static parameters only.
dynamicOnly				Prints dynamic parameters only.
solving				

Return value

void

Examples

```
// List the parameters for the selected character.
listParameters;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [hasParameter \(page 737\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [setParameter \(page 1118\)](#)

loadFile

Description

Summary

Load a native file (*.vdf or *.hdf). You are prompted for a filename.

Details

Use loadFile when you need to open an existing *.vdf file. This action will close the open scene, prompting you first to save it.

You can also use this command to import or merge non-native motion capture files into your existing scene.

Functional area

File handling

Command syntax

Syntax

```
loadFile ["filename1" "filename2" ...][-startTime integer] [-type string]
[-i] [-createSecondFig] [-importType string] [-range integer integer]
```

Arguments

Name	Type	Required	Comments
filename			String path name of file to be loaded (can be specified by the File Open or File Import dialog boxes or explicitly by name)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
startTime	1	integer	—	Force the incoming animation to start at the specified time.
type	1	string	—	Open the dialog with filter set to given extension (e.g. ".asf" sets extension filter to .asf)
i	0	—	—	Import from a non-native file. Importer is chosen by matching the file extension.
createSecondFig	0	—	—	Will automatically create a second clip with second figure data for each clip created upon import
importType	1	string	—	Arguments for this option are: ImportSelectedCreateNew - the default and does as indicated ImportOnlySelected - ignores any modules being imported that are not selected AlwaysCreateNew - forces importers to always create new modules CurrentClipCreateNew MergeAll MergeSelected
range	2	integer	—	—

Return value

boolean

Examples

```
loadFile -i -importType "mergeSelected";  
// This command prompts you to select a non-native file.  
// The selected file is merged into your existing scene.  
  
loadFile "C:/temp/myFile.vdf";  
// Loads a native VDF file
```

Additional information

Related commands

- [saveFile \(page 943\)](#)

loadScript

Description

Details

Use this command to load a script into the Script Editor.

Functional area

Interface

Command syntax

Syntax

```
loadScript ["scriptPath"]
```

Arguments

Name	Type	Required	Comments
scriptPath	string	yes	Path to the script file

Flags

None

Return value

void

Examples

```
loadScript ["C:\ShogunPostScripts\Pipelines\Process.hsl"]
```

loadWorkspaceString

Description

Summary

Load a preconfigured workspace layout

Details

Shogun Post has a number of preconfigured workspace layouts. These layouts include the visibility and position of the docking windows, as well as the types of panels that are open.

If you have your own set of workspace layouts you have created, you may load them using this command.

Functional area

Interface

Command syntax

Syntax

```
loadWorkspaceString "string"
```

Arguments

Name	Type	Required	Comments
string	string	Yes	Path to workspace file to load.

Flags

None

Return value

void

Examples

```
// Load one of your custom layouts  
loadWorkspaceString "C:/Program Files/Vicon/ShogunPost#.#/MyLayouts  
/TrackingLayout.xml";
```

Additional information

Related commands

■ [saveWorkspaceFile \(page 947\)](#)

makeRigid

Description

Summary

This command is used to enforce a rigid relationship among a selected group of markers for the entire file.

Details

In optical data capture, relationships among markers that should remain fixed, sometimes do not remain fixed. Noise in the system, occlusions, and physical collisions are some of the common factors that may contribute to the degradation of marker relationships (markers falling off the performer's waist, for example, or a head marker being lost as the performer reaches the edge of the capture volume).

The makeRigid command can be used to automatically correct these problems for multiple markers simultaneously. Restoration and/or strict enforcement of such relationships makes the subsequent conversion of marker data to skeletal data happen much more quickly and cleanly.

This command is used to enforce a rigid relationship among a selected group of markers for the entire file and overwrites existing data where it does not conform to the rigid relationship.

Optionally, you may create a new node (using the `-createRigidBody` option), which is defined as the averaged center of the selection set over time, with rotation keyframes applied to it by calculating the rotation of the group as a rigid unit.

Functional area

Data manipulators

Command syntax

Syntax

```
makeRigid [-createNewNodes] [-useCurrentFrame] [-createRigidBody string]
[-targetPrimary] [-parent] [-nonRigidTolerance float] [-
selectNonRigidKeys] [-noWarning]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>createNewNodes</code>	0	—	—	Apply rigid data to new nodes created from the currently selected nodes. Leaves source nodes untouched. The naming convention for the new nodes is: "Rigid_" + (name of source node)
<code>useCurrentFrame</code>	0	—	—	Use the current frame as the creation reference instead of automatically finding it. By default, Shogun Post automatically identifies the best frame in the playRange for the rigid body's creation reference.
<code>createRigidBody</code>	1	string	—	Simultaneously create a new rigid body from the selected markers.
<code>targetPrimary</code>	0	—	—	Puts rigid body information onto the primary selected object. Generally not used with the <code>createRigidBody</code> option.
<code>parent</code>	0	—	—	Parent rigidified nodes (new or existing) to newly created rigid body.
<code>nonRigidTolerance</code>	1	float	—	Sets the tolerance (measured in millimeters [float]) for qualifying source node keys as valid or invalid during rigid body calculations. Any keys on source nodes that fall outside this

Name	Flag arguments	Argument type	Exclusive to	Comments
				tolerance of being "rigid" with respect to the other source nodes, are ignored during rigid body calculations.
<code>selectNonRigidKeys</code>	0	—	—	Selects all keys that were ignored on the source nodes when calculating the rigid body. Can only be used in conjunction with <code>nonRigidTolerance</code> .
<code>noWarning</code>	0	—	—	Disables warning prints to the log that result from the command's execution.

Return value

void

Examples

```
// In this example, 4 markers corresponding to the chest are
// selected. In the next line, there is a rigid relationship
// enforced by makeRigid; the '-createRigidBody chest' option
// creates a new rigidBody object called "chest" which contains
// the translation and rotation information of the group as a whole;
// and finally, the four rigidified markers are parented (`-parent`)
// to the new rigidBody object. You may now operate on the group as a
// whole by editing the translation and rotation of the "chest" node.
select C7 T10 STRN CLAV;
makeRigid -createRigidBody chest -parent;
```

Additional information

Related commands

- [fillGaps \(page 389\)](#)
- [rigidBody \(page 937\)](#)

makeUnique

Description

Details

Fixes duplicate module names in the scene. This command fixes databases that contain modules with duplicate names. If duplicate names are encountered, a number is appended to the module name.

Functional area

Data editing

Command syntax

Syntax

```
makeUnique
```

Arguments

None

Flags

None

Return value

void

Examples

```
create Character Actor1 Actor1;  
// Fix the database as it has duplicate names  
makeUnique;
```

manipulator

Description

Summary

Establish and configure the current manipulator within the 3D views.

Details

The manipulator commands provide a flexible array of options for modifying object properties within the 3D views. For example, by using the options `-posTrack` and `-orientTrack`, you may establish the manipulator at the center of a group of objects. Subsequent modifications using the manipulator in these cases will take that location as the origin, modifying all members of the selection to together in the same local coordinate system.

Functional area

Interface

Command syntax

Syntax

```
manipulator [-posManip boolean] [-rotManip boolean] [-scaleManip boolean]
[-specialManip boolean] [-posTrack boolean] [-orientTrack boolean] [-
snap] [-follow boolean] [-selChangeFollow boolean] [-grid boolean] [-
posGridVal integer] [-rotGridVal integer] [-scaleGridVal integer] [-
logFeedback boolean] [-size float] [-inc float] [-dec float] [-space
string] [-editBasePose boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
posManip	1	boolean	rotManip, scaleManip	Show/hide the position manipulator, mutually exclusive with respect to [rotManip]
rotManip	1	boolean	posManip, scaleManip	Show/hide the rotation manipulator, mutually exclusive with respect to [posManip]
scaleManip	1	boolean	posManip, rotManip	Show/hide the scale manipulator,
specialManip	1	boolean		Show/hide the special manipulator,
posTrack	1	boolean	—	Set/unset any manipulator snapping or following to the center of the selected nodes position
orientTrack	1	boolean	—	Set/unset any manipulator snapping or following to the center of the selected nodes position
snap	0	—	—	Snaps manipulator to selected nodes
follow	1	boolean	—	Set/unset following of current selection
selChangeFollow	1	boolean	—	Set/unset following of current selection and any subsequent selection changes
grid	1	boolean	—	Set/unset manipulator grid snapping

Name	Flag arguments	Argument type	Exclusive to	Comments
posGridVal	1	integer	—	Set the pos manip grid snapping value
rotGridVal	1	integer	—	Set the rot manip grid snapping value
scaleGridVal	1	integer	—	—
logFeedback	1	boolean	—	Controls whether the feedback should be logged to the log file
size	1	float	inc, dec	Sets the manipulator size, in mm
inc	1	float	size, dec	—
dec	1	float	size, inc	—
alignEnd	0	—	alignWs, alignPole	—
alignWs	0	—	alignEnd, alignPole	—
alignPole	0	—	alignEnd, alignWs	—
space	1	string	—	—
editBasePose	1	boolean	—	—

Return value

void

Examples

```
select LFWT LMWT LBWT RFWT RMWT RBWT;
manipulator -rotManip on;
manipulator -orientTrack on -posTrack on;
// These commands select all members of a character's waist,
// and prep the manipulator's center of rotation to always
// remain in the middle of the selected group, enabling
// you to rotate the markers as a unit about their center.
// Changing the selection will not change the rotation manipulator,
// allowing you to rotate other markers about the average of the
// waist markers.
```

Additional information

Related commands

- [graphView \(page 730\)](#)
- [viewLayout \(page 1342\)](#)

markingMenu

Description

Summary

Set marking menu entry.

Functional area

System

Command syntax

Syntax

```
markingMenu zone mouse direction label command[-ctrl] [-shift] [-menuName  
string]
```

Arguments

See above syntax.

Flags

See above syntax.

Return value

Void

master

Description

Summary

Turn the DBS master on or off.

Details

This command allows you to enable or disable the DBS (Distributed Batch System) master. A master seat can select the files and parameters to batch process, control the execution of the batch process, and display the results.

For the master and slave to communicate, they must be on the same workgroup and port.

Functional area

Master/Slave

Command syntax

Syntax

```
master on/off/toggle[-workgroup string] [-port integer] [-ip string]
```

Arguments

Name	Type	Required	Comments
on/off/toggle	boolean	yes	Turn the master on or off

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
workgroup	1	string	—	Sets this master as part of the given processing workgroup. See <i>Details</i> above for information about processing workgroups.
port		integer		
ip		string		

Return value

void

Examples

```
// This command enables the DBS master and joins it as
// part of the "Farming" processing group.
master on -workgroup "Farming";
master toggle -workgroup "DBSWorkgroup" -port 5419 -ip "all";
```

Additional information

Related commands

■ [slave \(page 1203\)](#)

mcpExportOptions

Description

Summary

Sets export options for MCP export.

Details

Sets export options for exporting MCP files for the next time you export an MCP.

Functional area

File handling

Command syntax

Syntax

```
mcpExportOptions [-exportCameras boolean] [-exportSubjects boolean] [-  
exportCameraData boolean] [-exportSubjectMotion boolean]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

- [amcExportOptions \(page 135\)](#)
- [bvhExportOptions \(page 179\)](#)
- [c3dExportOptions \(page 181\)](#)
- [cpExportOptions \(page 232\)](#)
- [fbxImportOptions \(page 380\)](#)
- [fbxExportOptions \(page 376\)](#)
- [loadFile \(page 779\)](#)
- [mcplImportOptions \(page 798\)](#)
- [saveFile \(page 943\)](#)
- [x2dImportOptions \(page 1375\)](#)
- [xcplImportOptions \(page 1377\)](#)

mcplImportOptions

Description

Summary

Sets import options for MCP import.

Functional area

File handling

Command syntax

Syntax

```
mcplImportOptions [-importCal boolean] [-importSubjects boolean] [-importMotion boolean] [-importVideo boolean] [-moveVideo boolean]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)

-
- [c3dExportOptions \(page 181\)](#)
 - [cpExportOptions \(page 232\)](#)
 - [fbxImportOptions \(page 380\)](#)
 - [fbxExportOptions \(page 376\)](#)
 - [loadFile \(page 779\)](#)
 - [mcpExportOptions \(page 796\)](#)
 - [saveFile \(page 943\)](#)
 - [x2dImportOptions \(page 1375\)](#)
 - [xcplImportOptions \(page 1377\)](#)

messagePrompt

Description

Summary

Prompts the user with a window displaying a message and Yes/No/Ok/Cancel buttons.

Details

Prompts the user with a window displaying a message and a series of buttons.

Calling this command halts script execution until the user clicks on one of the buttons, or dismisses the window using the ESC key. Choices of button combinations are OK, OK/Cancel, Yes /No, and Yes/No/Cancel. The return value holds the code to the button that the user selected. If no button options are specified, an OK button only is displayed.

Here is a list of buttons and their codes/return values:

Button	Return Code
OK/Yes	1
No	0
Cancel	-1

Functional area

Interface

Command syntax

Syntax

```
messagePrompt "message" [-okCancel integer] [-yesNo integer] [-yesNoCancel integer]
```


Arguments

Name	Type	Required	Comments
message	string	yes	Message to display to the user.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
okCancel	1	integer	yesNo, yesNoCancel	Displays an OK and a Cancel button. Integer value specifies the "default" button.
yesNo	1	integer	okCancel, yesNoCancel	Displays a Yes and a No button. Integer value specifies the "default" button.
yesNoCancel	1	integer	okCancel, yesNo	Displays a Yes, a No, and a Cancel button. Integer value specifies the "default" button.

Return value

integer

Examples

```
// Prompt the user with a message and proceed according
// to the button that was clicked. Make No the default button
int $returnCode;
boolean $shouldSave;
$returnCode = `messagePrompt "Not all nodes are selected for export.\n\n Do you
still want to save?" -yesNoCancel 0`;
switch( $returnCode )
{
    case -1: // User clicked Cancel. Exit the script
        return;
    case 0:  // User clicked No. Don't save
        $shouldSave = false;
        break;
    case 1:  // User clicked Yes. Can proceed with save
        $shouldSave = true;
        break;
}
// Save if it's safe to save, according to the user.
$if( $shouldSave )
{
    saveFile -s;
}
```

Additional information

Related commands

■ [createWindow \(page 311\)](#)

moveClip

Description

Summary

Moves a clip around in the clip list.

Details

Moves the specified clip under another specified clip. Useful when trying to organize your clips in a particular required structure.

Functional area

Data manipulators

Command syntax

Syntax

```
moveClip "moveClip" ["insertAfterClip"] [-up] [-down]
```

Arguments

Name	Type	Required	Comments
moveClip	string	yes	The clip that you want to move
insertAfterClip	string	yes	The clip that you want the moveClip to be located under in the clip list.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
up	0	—	down	Moves the clip up from its current position in the clip list. Will error if the clip is already at the top of the clip list
down	0	—	up	Moves the clip down from its current position in the clip list. Will error if the clip is already at the bottom of the clip list

Return value

void

Examples

```
// This snippet creates 3 Clips then moves clip_C under clip_A
create Clip "clip_A" "clip_B" "clip_C";
moveClip clip_C clip_A;

// This will move clip_B to the top of the list
moveClip clip_B -up;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)

-
- [removeLayer \(page 905\)](#)
 - [resetClip \(page 931\)](#)
 - [selectClipObjects \(page 979\)](#)
 - [setActiveClip \(page 1017\)](#)
 - [setActiveClip \(page 1017\)](#)
 - [setActiveLayer \(page 1020\)](#)
 - [tileClips \(page 1311\)](#)

moveRotKeyToPreRotCmd

Description

Summary

Transforms

Functional area

Skeletal Solving

Command syntax

Syntax

```
moveRotKeyToPreRotCmd
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

■ [moveTransKeyToPreTransCmd \(page 807\)](#)

moveTransKeyToPreTransCmd

Description

Summary

Transforms

Functional area

Skeletal Solving

Command syntax

Syntax

```
moveTransKeyToPreTransCmd
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

■ [moveRotKeyToPreRotCmd \(page 806\)](#)

multiplyJointRange

Description

Summary

Multiplies the joint ranges across all axes for a joint.

Details

Enables you to multiply the joint ranges across all axes of a joint by a specified value. For example, if you have just calibrated and think the joint range for a specific joint needs to be altered for optimal labeling, instead of capturing a whole new ROM, you can just use multiplyJointRange to adjust the range for that particular joint.

Functional area

Skeletal solving

Command syntax

Syntax

```
multiplyJointRange multiplier [-onMod string]
```

Arguments

Name	Type	Required	Comments
multiplier			

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies the module on which the command operates.

Return value

void

Additional information

Related commands

■ [multiplyJointStiffness \(page 810\)](#)

multiplyJointStiffness

Description

Summary

Multiplies the joint stiffness of a particular joint by a specified value.

Details

Multiplies the internal covariance matrix and then updates the per-axis stiffness values of a specific joint to reflect the new matrix values. For example, if you have just calibrated and think the joint stiffness for a specific joint needs to be altered for optimal labeling, instead of capturing a whole new ROM, you can just use multiplyJointStiffness to adjust the stiffness for that joint.

Functional area

Skeletal solving

Command syntax

Syntax

```
multiplyJointStiffness multiplier[-onMod string]
```

Arguments

Name	Type	Required	Comments
multiplier			

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies the module on which the command operates.

Return value

void

Additional information

Related commands

■ [multiplyJointRange \(page 808\)](#)

newFile

Description

Summary

Clear the database and prepare for a new file.

Details

This command clears the database containing all data pertaining to the current scene. You are not prompted to save the current scene. This command is not undoable.

Functional area

File handling

Command syntax

Syntax

```
newFile [-resetScene] [-keepClipsAndCal] [-promptToSave]
```

Arguments

None

Flags

See above syntax.

Return value

void

Examples

```
newFile;  
// Database cleared.
```

Additional information

Related commands

- [loadFile \(page 779\)](#)
- [saveFile \(page 943\)](#)

normalize

Description

Summary

Normalizes a vector.

Details

Normalizes and returns a vector. Normalizing a vector is the process of making the vector "unit-length", or have a length of 1.

Functional area

Math

Command syntax

Syntax

```
normalize vector
```

Arguments

Name	Type	Required	Comments
vector	vector	yes	The vector to normalize.

Flags

None

Return value

vector

Returns a vector of length 1.0.

Examples

```
// Normalize a vector  
vector $vec = << 30, 60, 90 >>;  
$vec = `normalize $vec`;  
print $vec;
```

Additional information

Related commands

- | [cross \(page 316\)](#)
- | [dot \(page 363\)](#)
- | [getAngleTo \(page 447\)](#)
- | [getLength \(page 576\)](#)
- | [setLength \(page 316\)](#)

offlineCameraHealthCheck

Description

Summary

Highlights the state of each camera and identifies any cameras whose calibration may need to be repaired.

Details

During the reconstruction, if the the `-computeHealthStats` flag is used, Shogun Post calculates and analyzes the data, and then reports on the accuracy of each camera and the state of its calibration. `offlineCameraHealthCheck` uses this data to report cameras whose calibration may need repair.

For `offlineCameraHealthCheck` to succeed you **MUST** use the `-computeHealthStats` flag during reconstruction.

RMS distance (Root Mean Square distance)

This value reports the distance between the 2D image of the markers on the camera and the 3D reconstructions of those markers projected back to the cameras sensor. The value is reported in the **RMS Distance** column in the **Log** window. If the value reported is less than the error threshold, the status light in the **Perspective** view is green, which indicates that the calibration for the camera is good to use. If the value is greater than the error threshold, the status light is red, which indicates that the camera needs to be repaired or recalibrated. This typically happens if the camera has moved since the last calibration.

The threshold for determining good or bad health is set at 1.25 camera sensor pixels, where fewer than 1.25 pixels of error represents good health, and more than 1.25 pixels of error represents bad health.

Proportion unassigned

This value indicates the degree to which data from this camera was used to calculate 3D reconstructions. It is calculated as the proportion of unassigned camera rays from markers that were visible to the camera but not used during reconstruction. Typical values for a good calibration are below 40% (0.4). This value is reported as a percentage in the **Proportion Unassigned** column in the **Log** window. If the value reported is less than threshold, the status light in the **Perspective** view is green, which indicates that the calibration for the camera is good to use. If the value reported is greater than threshold, the status light is red, which indicates that the camera calibration is not good to use. This is typically due to one of the following causes:

- The camera calibration is good, but the camera is seeing some reflections. To check this, select the appropriate camera and switch to a **Cameras** view where you can see the 2D camera data. If you see unwanted reflections, remove the source of the reflections from the capture volume or mask it out.
- The camera calibration is bad, and the markers it is seeing are not being used to create 3D reconstructions at all. This typically happens if the camera has moved since the last calibration. This camera needs to be repaired or recalibrated.

Shogun Post displays two status lights under each camera in the **Perspective** view (use option `-showCameraHealth` to turn on/off). For both status lights, the color indicates:

- Green: The camera calibration is good and no other issues have been detected.
- Red: There is a potential problem that needs to be reviewed.
- Default color: No data was acquired for that camera to provide status information.

Functional area

Data retrieval

Command syntax

Syntax

```
offlineCameraHealthCheck [-rmsThreshold float] [-proportionThreshold float] [-showCameraHealth boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>rmsThreshold</code>	1	float	—	Set the threshold value of RMS distance (pixels, default value is 1.25)
<code>proportionThreshold</code>	1	float	—	Set the threshold value of proportion of unassigned camera rays from markers (percent, default value is 0.40)

Name	Flag arguments	Argument type	Exclusive to	Comments
showCameraHealth	1	boolean	—	Turn on/off/toggle status lights for the cameras in the Perspective view

Return value

string array

The names of the cameras whose calibration may need to be repaired

Examples

```
// Load a file with a known camera which has moved and reconstruct
reconstruct -currentFrame -computeHealthStats;

// Check cameras' health and print cameras whose calibration may
// need to be repaired
string $cameras[] = `offlineCameraHealthCheck -RMSThreshold 1.25 -
proportionThreshold 0.4 -showCameraHealth on`;
print $cameras;
```

Additional information

Related commands

■ [reconstruct \(page 882\)](#)

offlineDataProvider

Description

Summary

Start/Stop offline data provider.

Functional area

Data Streaming

Command syntax

Syntax

```
offlineDataProvider on/off/toggle
```

Arguments

Name	Type	Required	Comments
on/off/toggle			

Flags

None

Return value

void

offsetClips

Description

Summary

Offsets clips in time

Details

Enables you to offset clips in time by a specified number of frames. The offset is additive, that is, it will add the offset to the existing value of the clip's `Clip_Offset` attribute.

Functional area

Data manipulators

Command syntax

Syntax

```
offsetClips "offset timecode" ["clip" ...."clip"]
```

Arguments

Name	Type	Required	Comments
offset timecode	integer	yes	The number of frames to offset the clip by
clip	string	no	The clip(s) to offset. May specify multiple clips by name or specify an array of clips. If no clips are specified, then the active clip is offset.

Flags

None

Return value

void

Examples

```
// Offset the active clip 50 frames back in time  
offsetClips -50;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

openEclipseDatabase

Description

Summary

Opens the specified Eclipse database.

Details

Functional area

Data management

Command syntax

Syntax

```
openEclipseDatabase ["database enf file path"]
```

Arguments

Name	Type	Required	Comments
database enf file path			

Flags

None

Return value

void

Additional information

Related commands

- [getEclipseActiveTrial \(page 515\)](#)
- [getEclipseAssociatedCalibration \(page 517\)](#)
- [getEclipseAssociatedSubjects \(page 519\)](#)
- [getEclipseMarkedTrials \(page 521\)](#)
- [refreshEclipse \(page 895\)](#)

pack

Description

Summary

Pack marker data

Details

This command will pack unlabeled trajectories. Packing is the process of copying all of the data from one trajectory onto an "empty" area on another trajectory. This is useful when you have many small trajectories and are saving out C3D files, as C3D files are very inefficient when saving out files with many small trajectories.

Use packing on gap-less trajectories only, or when you don't wish to preserve the gaps in trajectories, because the unpacking process will treat each contiguous key range as a new trajectory.

Functional area

Data editing

Command syntax

Syntax

```
pack [-minGap integer] [-minDistance float] [-marker string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
minGap	1	integer	—	Specifies the minimum amount of frames to leave between two trajectories. The default value is 1 if not specified.
minDistance	1	float	—	Specifies the minimum distance, in millimeters, between the end of one trajectory and the start of another trajectory to allow them to be packed next to each other. The default value is 0 millimeters if not specified.
marker	1	string	—	If specified, only this trajectory will be packed. Otherwise all selected unlabeled trajectories will be packed.

Return value

void

Examples

```
// This selects and packs unlabeled trajectories in the scene and
// ensures that there is a 1 frame gap between each packed trajectory,
// and that each trajectory is 100 mm away from the end of the previous
// trajectory
selectByName "_" -type Marker;
pack -minGap 1 -minDistance 100;

// This packs only the unlabeled trajectory _5 onto one of the
// selected unlabeled trajectories.
// It uses default values for the rest of the options
pack -marker _5;
```

Additional information

Related commands

■ [unpack \(page 1329\)](#)

parent

Description

Summary

Make a node a child of another node.

Details

The parent command allows you to build hierarchies among your scene objects. The transformations of an object remain unchanged after the parent operation unless the -adjust option is used. Therefore, if the transformations of the new parent are not identical to the previous parent, the parented object will be in a new absolute workspace location subsequent to the parent operation.

Functional area

Data manipulators

Command syntax

Syntax

```
parent [-adjust] [-showProgress] [-ranges] [-adjustOnly]
```

Arguments

Implied based on currently selected objects. The primary selected object will become the parent of all other selected objects.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
adjust	0	—	adjustOnly	Recalculates the transform of each child so as to account for the transform of the parent
showProgress	0	—	—	—
ranges	0	—	—	—
adjustOnly	0	—	adjust	—

Return value

void

Examples

```
select RHEL RMT5 RMT1 RTOE RANK;
parent -adjust;
// These commands parent the all markers belonging to the right
// foot besides the ankle, to the ankle. After you run this command, you
// can operate on the markers as a unit by transforming just the RANK
// node.
```

Additional information

Related commands

■ [unparent \(page 1331\)](#)

pasteKeys

Description

Summary

Pastes keys from the clipboard onto the selected channels of the selected module at the current frame

Details

The pasteKeys command may only be used under these conditions:

- Exactly one module is selected
- A single channel selection is made within only one Property of the module

Keys are then pasted starting at the current frame, unless the useFrame option is used to specify a different starting frame.

Functional area

Data editing

Command syntax

Syntax

```
pasteKeys [-useFrame integer]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
useFrame	1	integer	—	Frame number at which to start pasting. If this option is not specified, the paste operation will begin at the current frame.

Return value

void

Examples

```
// Select the X Rotation channel of the module right_foot, and paste
// the keys within the clipboard to those channels, beginning at the
// current frame.
select right_foot;
selectProperty "RotationX";
pasteKeys;
```

Additional information

Related commands

- [copyKeys \(page 224\)](#)
- [cutKeys \(page 318\)](#)

pathExists

Description

Summary

Determine whether a file or folder exists

Details

Determine whether a file or folder exists on the file system.

Note that all file and folder paths in Shogun Post HSL scripting use forward slashes instead of back-slashes.

Functional area

System

Command syntax

Syntax

```
pathExists "folderPath"
```

Arguments

Name	Type	Required	Comments
folderPath	string	yes	The name of the file or folder to search for.

Flags

None

Return value

boolean

Returns true if the file or folder path specified exists on the file system.

Examples

```
// See if this file exists on our disk
boolean $itExists = `pathExists "C:/temp/test.txt"`;
print $itExists;
```

play

Description

Summary

Play animation

Details

The play command initiates playback of the animation in the scene.

Functional area

Playback control

Command syntax

Syntax

```
play [-b] [-t]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
b	0	—	—	Backward
t	0	—	—	Toggle play/stop

Return value

void

Examples

```
play -b;  
// Play the scene animation backwards.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

playOptions

Description

Summary

Reset the play options to their defaults.

Details

The playOptions command allows for the editing of scene details relating to animation playback.

Functional area

Playback control

Command syntax

Syntax

```
playOptions [-min float] [-max float] [-animMin float] [-animMax float] [-fps float] [-playSpeed string] [-loop boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
min	1	float	—	Integer for play range minimum
max	1	float	—	Integer for play range maximum
animMin	1	float	—	Integer for animation range minimum

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>animMax</code>	1	float	—	Integer for animation range maximum
<code>fps</code>	1	float	—	Integer to set frame rate (frames per second)
<code>playSpeed</code>	1	string	—	Takes a string argument of the same strings found in the list box on the time bar (i.e. "1/8x", "1x". etc)
<code>loop</code>	1	boolean	—	Loop playback. Not used with [bidirectional]

Return value

void

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

playRange

Description

Summary

Set play range to include the frame range specified.

Details

Use this command in the **Script Editor** to set the playRange of your scene to a subset of the current range.

playRange is very useful for trimming the working range of your animation file without losing any of the data. When you reset a playRange, the data that lies outside of the new range will still remain in the file if saved as a *.vdf*. Upon export to another format, only the playRange is exported.

The playRange command is not undoable.

SMPTE times must be surrounded by quotes

Functional area

Playback control

Command syntax

Syntax

```
playRange inTime outTime[-r] [-start] [-end] [-save] [-restore]
```

Arguments

Name	Type	Required	Comments
inTime		yes	Start frame integer , required unless -end option is specified
outTime		yes	End frame integer, required unless the -start option is specified

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
r	0	—	—	Set range relative to start/end of current play range
start	0	—	end	Sets the playRange start to the current frame
end	0	—	start	Sets the playRange end to the current frame
save	0	—	restore	Save current play range before applying play range changes. Can be restored by next playRange -restore call.
restore	0	—	save	Restore the last saved play range before applying play range changes.

Return value

void

Examples

```
playRange -save;
playRange 340 345;
playRange -restore;
// This sequence of commands saves the current play range,
// then sets it to a new range; enabling you to restore
// the original range at any time.

// set start of play Range to 50
playRange -start 50;

// set end of play Range to 200
playRange -end 200;
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

playSound

Description

Summary

Plays a system sound or a .wav file.

Details

Plays the default system beep, or a .wav file if one is specified.

Note that the sound plays asynchronously. i.e. script execution continues as the sound plays.

Functional area

System

Command syntax

Syntax

```
playSound [-file string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Plays the specified .wav file, instead of the default system beep.

Return value

void

Examples

```
// Sound the system beep to indicate the end of a script  
// ... long script ...  
playSound;
```


print

Description

Summary

Print the contents of the specified variables to the **Log**.

Details

The print command will print the contents of any variable(s) to the feedback line. Each value (and array element) will get its own log entry. If you specify `-warning` or `-error`, the value printed for each entry will have the words "Warning: " or "ERROR: " prepended to it, respectively, and will draw with an alternate background color on the log line of the status bar.

Functional area

System

Command syntax

Syntax

```
print "stringToPrint" [-warning] [-error]
```

Arguments

Name	Type	Required	Comments
stringToPrint	any	yes	<p>This argument can be any data type. If it is an array type, each array value will be printed to the command log in its own line.</p> <p>You may specify any number of variables/values to print.</p>

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
warning	0	—	error	Marks each log entry as a warning, which will prepend the word "Warning: " to the entry as well as use a different background color in the Log line of the status bar.
error	0	—	warning	Marks each log entry as an error, which will prepend the word "ERROR: " to the entry as well as use a different background color in the Log line of the status bar.

Return value

void

Examples

```
// Demonstrate the versatility of the print command
string $str[2];
vector $vec = << 1, 2, 3 >>;
$str[0] = "test 1";
$str[1] = "test 2";
print $str;
print $vec;
print -error "Something very wrong has happened";
```

processFile

Description

Summary

Command used by batch processing: the options match those in the **Batching** window.

Functional area

System

Command syntax

Syntax

```
processFile "filePath"[-seedFile string] [-noExport] [-appendage string]
[-replaceAppendage] [-outputDir string] [-script string] [-scriptFile
string] [-rate string] [-type string] [-log string] [-shotIds integer
array] [-shotIdTypes integer array] [-shotCharacters string array] [-
isMasterFile]
```

Arguments

Name	Type	Required	Comments
filePath	string	yes	The name of the file to search for.

Flags

See above syntax.

Return value

Void

progressBar

Description

Summary

Controls the progress bar displayed on Shogun Post's status bar.

Details

To accommodate multiple, nested, and both user and internally created progress bars, Shogun Post uses the concept of an "operation stack". An operation is any arbitrary script section whose progress needs to be displayed as feedback to the user during script execution.

An operation is started by calling the command with the `-push` flag, which will create a new operation, and "push" it to the top of the stack. All subsequent calls to `progressBar` will act on this operation.

If the `-push` flag is not specified at the start of an operation, the command will error, because there will be no operation to act on. Conversely, to finish an operation, call the command with the `-pop` flag, which will remove the operation at the top of the stack. If `-pop` is called without a preceding and matching `-push`, the command will fail.

The concept of a stack allows multiple operations to be nested by pushing another operation onto the stack before popping the first one off. For example:

```
progressBar -push -text "Operation A";// Pushes Operation A
progressBar -push -text "Operation B";// Pushes Operation B
progressBar -pop; // Pops Operation B
progressBar -pop; // Pops Operation A
```

Besides giving feedback to the user during lengthy operations, adding `progressBar` commands to your script also enables the user to cancel execution of the script using the **Cancel** button on the status bar. The **Cancel** button is disabled otherwise.

Functional area

Interface

Command syntax

Syntax

```
progressBar [-push] [-pop] [-range integer integer] [-pos integer] [-step] [-text string] [-noCancel]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
push	0	—	pop	Pushes a new operation onto the operation stack. The first call of an operation must use this flag.
pop	0	—	push, range, pos, step, text	Pops an operation off of the operation stack.
range	2	integer	pop	Sets the range of the operation at the top of the operation stack. The two integers specify the start and end of the range. If the operation stack is empty, the command fails.
pos	1	integer	pop, step	Sets the position of the operation at the top of the operation stack. If the operation stack is empty, the command fails.
step	0	—	pop, pos	Increments the position of the operation at the top of the operation stack by one. If the stack is empty, the command fails.
text	1	string	pop	Sets the text of the operation at the top of the operation stack. If the operation stack is empty, the command fails.

Name	Flag arguments	Argument type	Exclusive to	Comments
noCancel	0	—	—	Disables the Cancel button so that the operation at the top of the operation stack cannot be canceled by the user.

Return value

void

Examples

```
// Demonstrate range setting, stepping, and nesting of operations
int $i, $j, $timeKiller, $maxI = 20, $maxJ = 2000;

// Push a new operation onto the stack.
progressBar -push -range 1 $maxI -text "Outer";

for( $i = 0;
    $i < $maxI;
    $i += 1 )
{
    // This increments the "Outer" operation
    progressBar -step;
    // Kill some time so the stepping effect is visible.
    $timeKiller = 0;
    while( $timeKiller < 5000 )
    {
        $timeKiller += 1;
    }
    // Push another operation onto the stack, which nests
    // this operation in the "Outer" one.
    progressBar -push -range 1 $maxJ -text "Inner";
    for( $j = 0;
        $j < $maxJ;
        $j += 1 )
    {
        // This increments the "Inner" operation
        progressBar -step -text string( $j );
    }
    // This pops the "Inner" operation off the stack.
    progressBar -pop;
}
// Finally, this pops the "Outer" operation off the stack.
progressBar -pop;
```

Additional information

Related commands

■ [createWindow \(page 311\)](#)

python

Description

Enables you to run a specified Python command string.

You can call a Python script from HSL in the same way as a native script. Python scripts called from Shogun Post can also be given arguments.

Note that you cannot get a return value back from Python when called in this way, and you cannot step into a Python script in the debugger.

Functional area

System

Command syntax

Syntax

```
python "command"
```

Arguments

Name	Type	Required	Comments
"command"	string	yes	The python command string to execute

Flags

None

Return value

void

Examples

```
python "shogunPost.NewScene()" ;
```


quickPost

Description

Summary

Executes the quick post operation that processes an x2d offline at the specified processing level.

Details

Functional area

Data editing

Command syntax

Syntax

```
quickPost processingLevel [-ranges] [-currentFrame] [resetScene]
```

Arguments

Name	Type	Required	Comments
processingLevel		yes	Type of processing required ("reconstruct", "label", "solve", "labelocclusionfix", "occlusionfixsolve", "labelsolve", or "labelocclusionfixsolve")

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges				Quick post operation processes the specified range(s) of the current capture
currentFrame				Quick post operation processes the current frame of the current capture
resetScene				Resets the current scene, deleting all animation data and unlabeled markers from the scene, but leaving MX devices and VSTs/VSKs untouched (equivalent of Reset Scene option in the Processing panel).

Return value

void

Example

```
// Reconstruct take for the current frame while resetting the scene.
// To process takes, replace "Reconstruct" with required processing
// type.
quickPost "Reconstruct" -currentFrame -resetScene;
```

Additional information

Related commands

- [quickPostLabelOptions \(page 851\)](#)
- [quickPostOcclusionFixOptions \(page 853\)](#)
- [quickPostSolveOptions \(page 854\)](#)

quickPostLabelOptions

Description

Summary

Sets the quick post processing level (Label).

Details

Functional area

Data editing

Command syntax

Syntax

```
quickPostLabelOptions [-useLabelingClusters boolean] [-
labelCompletenessEntranceThreshold float] [-useRobustBooting boolean] [-
bootingQuality float] [-bootingVersusTracking float] [-trackingQuality
float] [-smoothingFactor float] [-enforceJointRanges boolean] [-
jointRangesSlack float] [singlePass boolean] [clearExistingLabels
boolean] [-numThreads float] [-reset] [-fromRTK] [-toRTK]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

- [quickPost \(page 849\)](#)
- [quickPostOcclusionFixOptions \(page 853\)](#)
- [quickPostSolveOptions \(page 854\)](#)

quickPostOcclusionFixOptions

Description

Sets the quick post processing level.

Functional area

Data editing

Command syntax

Syntax

```
quickPostOcclusionFixOptions [-fixOcclusionsEnabled boolean] [-  
applyFixedMarkers boolean] [backup -boolean] [-markerSmoothing float] [-  
dataFidelity float] [-transitionTime float] [-reset] [-fromRTK] [-toRTK]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

- [quickPost \(page 849\)](#)
- [quickPostLabelOptions \(page 851\)](#)
- [quickPostSolveOptions \(page 854\)](#)

quickPostSolveOptions

Description

Sets the quick post processing level (Solve).

Functional area

Data editing

Command syntax

Syntax

```
quickPostSolveOptions [-priorImportance float] [-meanPoseRatio float] [-  
extrapolateFingers boolean] [-fingerMultiplier float] [-numThreads  
string] [-reset] [-fromRTK] [-toRTK]
```

Arguments

None

Flags

See above syntax.

Return value

void

Additional information

Related commands

- | [quickPost \(page 849\)](#)
- | [quickPostLabelOptions \(page 851\)](#)
- | [quickPostOcclusionFixOptions \(page 853\)](#)

readBool

Description

Summary

Read Boolean value from a file.

Details

Use to read a Boolean value from a file. Command will fail if at the end of the file. The following is a listing of the different cast types.

Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means the value will always be positive.

Cast	Description
c	Reads a signed character, or a signed 8 bit (1 byte) value, and converts it to a Boolean
uc	Reads an unsigned character, or a unsigned 8 bit (1 byte) value, and converts it to a Boolean
s	Reads a signed short integer, or a signed 16 bit (2 bytes) value, and converts it to a Boolean
us	Reads an unsigned short integer, or an unsigned 16 bit (2 bytes) value, and converts it to a Boolean
i	Reads a signed integer, or a signed 32 bit (4 bytes) value, and converts it to a Boolean
ui	Reads an unsigned integer, or an unsigned 32 bit (4 bytes) value, and converts it to a Boolean
f	Reads a floating point value, a 32 bit (4 byte) value, and converts it to a Boolean

Functional area

Disk I/O

Command syntax

Syntax

```
readBool fileID [-cast string]
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast from. Possible types are listed below. Note that data can be lost when casting data.

Return value

boolean

Examples

```
//Read example
boolean $val;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "r"`;

// Read the boolean value
$val = `readBool $fileID`;

fileClose $fileID; //make sure to close the file

// Display the value
print $val;
```

Additional information

Related commands

- | [readFloat \(page 858\)](#)
- | [readInt \(page 861\)](#)
- | [readRot \(page 866\)](#)
- | [readString \(page 869\)](#)
- | [readVec \(page 874\)](#)

readFloat

Description

Summary

Read floating point value from a file.

Details

Use to read a floating point value from a file. Command will fail if at the end of the file.

The following is a listing of the different cast types.

Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
c	Reads a signed character, or a signed 8 bit (1 byte) value, and converts it to a float
uc	Reads an unsigned character, or a unsigned 8 bit (1 byte) value, and converts it to a float
s	Reads a signed short integer, or a signed 16 bit (2 bytes) value, and converts it to a float
us	Reads an unsigned short integer, or an unsigned 16 bit (2 bytes) value, and converts it to a float
i	Reads a signed integer, or a signed 32 bit (4 bytes) value, and converts it to a float
ui	Reads an unsigned integer, or an unsigned 32 bit (4 bytes) value, and converts it to a float
f	Reads a floating point value, a 32 bit (4 byte) value, and converts it to a float

Functional area

Disk I/O

Command syntax

Syntax

```
readFloat fileID [-cast string]
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast from. Possible types are listed above. Note that data can be lost when casting data.

Return value

float

Examples

```
//Read example
float $val;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "r"`;

// Read the boolean value
$val = `readFloat $fileID`;

fileClose $fileID; //make sure to close the file

// Display the value
print $val;
```

Additional information

Related commands

- | [readBool \(page 855\)](#)
- | [readInt \(page 861\)](#)
- | [readRot \(page 866\)](#)
- | [readString \(page 869\)](#)
- | [readVec \(page 874\)](#)

readInt

Description

Summary

Read integer value from a file

Details

Use to read an integer value from a file. Command will fail if at the end of the file.

The following is a listing of the different cast types.

Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
c	Reads a signed character, or a signed 8 bit (1 byte) value, and converts it to an integer
uc	Reads an unsigned character, or a unsigned 8 bit (1 byte) value, and converts it to an integer
s	Reads a signed short integer, or a signed 16 bit (2 bytes) value, and converts it to an integer
us	Reads an unsigned short integer, or an unsigned 16 bit (2 bytes) value, and converts it to an integer
i	Reads a signed integer, or a signed 32 bit (4 bytes) value, and converts it to an integer
ui	Reads an unsigned integer, or an unsigned 32 bit (4 bytes) value, and converts it to an integer
f	Reads a floating point value, a 32 bit (4 byte) value, and converts it to an integer

Functional area

Disk I/O

Command syntax

Syntax

```
readInt fileID [-cast string]
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast from. Possible types are listed above. Note that data can be lost when casting data.

Return value

integer

Examples

```
//Read example
int $val;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "r"`;

// Read the boolean value
$val = `readInt $fileID`;

fileClose $fileID; //make sure to close the file
// Display the value
print $val;
```

Additional information

Related commands

- | [readBool \(page 855\)](#)
- | [readFloat \(page 858\)](#)
- | [readRot \(page 866\)](#)
- | [readString \(page 869\)](#)
- | [readVec \(page 874\)](#)

readLine

Description

Summary

Reads a whole line of text from a file

Details

Use to read a whole line of text from a file. Will read up until a newline character is encountered, or until 1024 characters have been read, whichever comes first.

Command will fail if at the end of the file.

Functional area

Disk I/O

Command syntax

Syntax

```
readLine fileID
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

None

Return value

string

Examples

```
string $line;
int $fileID;

// Open a sample text file
$fileID = `fileOpen "c:/samplefile.txt" "r"`;

// Keep reading in lines from the file
while( `isEndOfFile $fileID` == false )
{
    $line = `readLine $fileID`;
    // Do something with line
    print $line;
}
```

Additional information

Related commands

- [readString \(page 869\)](#)
- [readToken \(page 872\)](#)

readRot

Description

Summary

Read rotation from a file

Details

Use to read an rotation from a file. Command will fail if at the end of the file.

The following is a listing of the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
c	Reads a series of signed characters, or a signed 8 bit (1 byte) values, and converts them to floats
uc	Reads a series of unsigned characters, or a unsigned 8 bit (1 byte) values, and converts them to floats
s	Reads a series of signed short integers, or a signed 16 bit (2 bytes) values, and converts them to floats
us	Reads a series of unsigned short integers, or an unsigned 16 bit (2 bytes) values, and converts them to floats
i	Reads a series of signed integers, or a signed 32 bit (4 bytes) values, and converts them to floats
ui	Reads a series of unsigned integers, or an unsigned 32 bit (4 bytes) values, and converts them to floats
f	Reads a series of floating point values, a 32 bit (4 byte) values, and converts them to floats

Functional area

Disk I/O

Command syntax

Syntax

```
readRot fileID [-delim string] [-quat] [-rotOrder string] [-cast string]
[-rad]
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
delim	1	string	—	Delimiter to expect between rotation channel values. Cannot be used with binary files. Usually ",", or "\t", etc.
quat	0	—	rotOrder	Specifies that the rotation should be expected as a quaternion (four float values) rather than an Euler angle (three float values). Can not be used with -rotOrder option
rotOrder	1	string	quat	Specifies the rotation order that the rotation will be expected in. Can not be used with -quat option (possible values are XYZ, XZY, YXZ, YZX, ZXY, and ZYX)
cast	1	string	—	Data type to cast to. Possible types are listed below. Note that data can be lost when casting data.
rad	0	—	quat	—

Return value

vector

Examples

```
//Read Rot Example
vector $rot;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "r"`;

// Read the rotation as an Euler angle
// The -delims flag assumes you are reading from a text file
$rot = `readRot $fileID -delim ","`;
// Or, read the rotation as a quaternion
// The -delims flag assumes you are reading from a text file
$rot = `readRot $fileID -quat -delim "\t"`;
fileClose $fileID;
// Display the value
print $rot;
```

Additional information

Related commands

- | [readBool \(page 855\)](#)
- | [readFloat \(page 858\)](#)
- | [readInt \(page 861\)](#)
- | [readString \(page 869\)](#)
- | [readVec \(page 874\)](#)

readString

Description

Summary

Reads a string from a file

Details

Use to read a string from a file. Command will fail if at the end of the file.

The following is a listing of the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Casting doesn't make sense when reading strings, but the option is still there for you.

Cast	Description
c	Reads each string character as a signed character, or a signed 8 bit (1 byte) value
uc	Reads each string character as an unsigned character, or a unsigned 8 bit (1 byte) value
s	Reads each string character as a signed short integer, or a signed 16 bit (2 bytes) value
us	Reads each string character as an unsigned short integer, or an unsigned 16 bit (2 bytes) value
i	Reads each string character as a signed integer, or a signed 32 bit (4 bytes) value
ui	Reads each string character as an unsigned integer, or an unsigned 32 bit (4 bytes) value
f	Reads each string character as a floating point value, or a 32 bit (4 byte) value

Functional area

Disk I/O

Command syntax

Syntax

```
readString fileID length [-cast string]
```

Arguments

Name	Type	Required	Comments
length	int		Length of the string to read from the file. Must be a positive value. Will read in characters up to the newline character, or length, whichever comes first. Will not return the newline character. Maximum value is 1024
fileID	int		ID of file previously opened with fileOpen (page 387) .

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast to. Possible types are listed below. Note that data can be lost when casting data.

Return value

string

Examples

```
string $str;
int $length;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "r"`;
// First read the string length
$length = `readInt $fileID` + 1;
// Now read the string
$str = `readString $fileID $length`;
```

```
fileClose $fileID; //close the file
// Print out the value
print $str;
```

Additional information

Related commands

- | [readBool \(page 855\)](#)
- | [readFloat \(page 858\)](#)
- | [readInt \(page 861\)](#)
- | [readRot \(page 866\)](#)
- | [readVec \(page 874\)](#)

readToken

Description

Summary

Reads a string token from a file.

Details

Use to read a string token from a file. A string token is simply a series of characters, separated by any number of delimiters. For example, the sentence "ShogunPost is for editing" is composed of the tokens "ShogunPost", "is", "for", and "editing", assuming the delimiter is a space. You can "tokenize" the contents of a file for easy parsing.

Note that this command can only be used with text files (files opened without the -b flag). Command will fail if at the end of the file.

Functional area

Disk I/O

Command syntax

Syntax

```
readToken fileID [-delims string]
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
delims	1	string	—	Delimiters to expect between tokens. Default is space, tab, or newline.

Return value

string

Examples

```
string $token;
int $fileID;
// Open a sample text file
$fileID = `fileOpen "c:/samplefile.txt" "r"`;
// Keep reading in tokens from the file
while( `isEndOfFile $fileID` == false )
{
    $token = `readToken $fileID`;
    // Do something with token
    print $token;
}
```

Additional information

Related commands

- [readLine \(page 864\)](#)
- [readString \(page 869\)](#)

readVec

Description

Summary

Read vector from a file

Details

Use to read a vector from a file. Command will fail if at the end of the file.

The following is a listing of the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
c	Reads a series of signed characters, or a signed 8 bit (1 byte) values, and converts them to floats
uc	Reads a series of unsigned characters, or a unsigned 8 bit (1 byte) values, and converts them to floats
s	Reads a series of signed short integers, or a signed 16 bit (2 bytes) values, and converts them to floats
us	Reads a series of unsigned short integers, or an unsigned 16 bit (2 bytes) values, and converts them to floats
i	Reads a series of signed integers, or a signed 32 bit (4 bytes) values, and converts them to floats
ui	Reads a series of unsigned integers, or an unsigned 32 bit (4 bytes) values, and converts them to floats
f	Reads a series of floating point values, a 32 bit (4 byte) values, and converts them to floats

Functional area

Disk I/O

Command syntax

Syntax

```
readVec fileID[-delim string] [-cast string]
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
delim	1	string	—	Delimiter to expect between vector channel values. Cannot be used with binary files. Usually ",", or "\t", etc.
cast	1	string	—	Data type to cast to. Possible types are listed below. Note that data can be lost when casting data.

Return value

vector

Examples

```
vector $vec;  
int $fileID;  
int $fileID = `fileOpen "C:/FileTesting.txt" "r"`;  
// Read the vector  
// The -delims flag assumes you are reading from a text file  
$vec = `readVec $fileID -delim ","`;  
fileClose $fileID;  
  
// Display the value  
print $vec;
```

Additional information

Related commands

- | [readBool \(page 855\)](#)
- | [readFloat \(page 858\)](#)
- | [readInt \(page 861\)](#)
- | [readRot \(page 866\)](#)
- | [readString \(page 869\)](#)

readWord

Description

Summary

Reads a string from a file

Details

Use to read a word from a file. Command will fail if at the end of the file.

Functional area

Disk I/O

Command syntax

Syntax

```
readWord fileID
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

None

Return value

string

Examples

```
string $str;  
int $fileID;  
int $fileID = `fileOpen "C:/FileTesting.txt" "r"`;  
// Now read the string  
$str = `readWord $fileID`;  
// Print out the value  
print $str;  
fileClose $fileID;
```

Additional information

Related commands

- | [readBool \(page 855\)](#)
- | [readFloat \(page 858\)](#)
- | [readInt \(page 861\)](#)
- | [readRot \(page 866\)](#)
- | [readString \(page 869\)](#)
- | [readVec \(page 874\)](#)

reassemble

Description

Summary

Reassemble .c3d files

Details

This command is used to reassemble a set of .c3d files generated by splitting the reconstruction of a take into smaller chunks (see [reconstructChunk \(page 884\)](#)). For example, a long capture session of 10,000 frames can be reconstructed on five different machines in parallel, each machine processing a different set of 2000 frames. The resulting files can then be reassembled into a single .c3d file using this command.

By default, this command reparents the markers to the first character in the scene, and deletes any other characters in the scene.

Functional area

Data editing

Command syntax

Syntax

```
reassemble "fileArray"[-tolerance integer] [-minGap integer] [-packEach]  
[-vLabel] [-reparent boolean]
```

Arguments

Name	Type	Required	Comments
fileArray	string	yes	List of files to be reassembled

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
tolerance	1	integer	—	Maximum number of frames that may overlap between source files when reassembling. The default is 25 frames.
minGap	1	integer	—	The minimum number of frames needed between two markers' data ranges when packing markers.
packEach	0	—	vLabel	Use this option to pack markers during the read operation on each file, rather than after all the markers have been read in.
vLabel	0	—	packEach	Set this option to use velocity labeling, which blends transitions in marker positions between the different file data.
reparent	0	boolean	—	Set to true to reparent all markers to first character (deleting all other characters in the scene).

Return value

void

Examples

```
// Divide up a long reconstruction
reconstructChunk "Take1.x2d" 0 2005 -fileName "Chunk1.c3d";
reconstructChunk "Take1.x2d" 2000 4005 -fileName "Chunk2.c3d";
reconstructChunk "Take1.x2d" 4000 6005 -fileName "Chunk3.c3d";
reconstructChunk "Take1.x2d" 6000 8005 -fileName "Chunk4.c3d";
reconstructChunk "Take1.x2d" 8000 10000 -fileName "Chunk5.c3d";
reassemble "Chunk1.c3d" "Chunk2.c3d" "Chunk3.c3d" "Chunk4.c3d" "Chunk5.c3d";
```

Additional information

Related commands

- [reconstructChunk \(page 884\)](#)

reconstruct

Description

Summary

Reconstruct 2D camera data into 3D marker data

Details

Reconstructs a capture over the entire play range unless the `-ranges` or `-currentFrame` flag is used. The [reconstructOptions \(page 887\)](#) command can be used to set reconstruction options.

Functional area

Data editing

Command syntax

Syntax

```
reconstruct [-ranges] [-currentFrame] [-computeHealthStats]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	0	—	—	Reconstructs only over the selected ranges of time.
currentFrame	0	—	—	Reconstructs only the current frame.
computeHealthStats				

Return value

void

Examples

```
// Reconstruct the entire play range.
reconstruct;
```

Additional information

Related commands

- [reconstructChunk \(page 884\)](#)
- [reconstructOptions \(page 887\)](#)

reconstructChunk

Description

Summary

Reconstructs a portion of an X2D file.

Details

Reconstructs a portion of an X2D file without the file and cameras needing to already exist in the scene. Options allow calibration file to be specified, scripts to be run before or after calibration, and various export options to be set.

Functional area

Data editing

Command syntax

Syntax

```
reconstructChunk "pathToX2D" firstFrame lastFrame [-cal string] [-
fileName string] [-appendage string] [-outputDir string] [-script string]
[-scriptFile string] [-log string]
```

Arguments

Name	Type	Required	Comments
pathToX2D			Path to required calibration file.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cal	1	string	—	Specifies the calibration file to use during reconstruction.
fileName	1	string	—	Specifies the name of the exported file.
appendage	1	string	—	Specifies a text string to append to the output file.
outputDir	1	string	—	Specifies the path to export the file to.
script	1	string	—	Specifies script code to run after reconstructing.
scriptFile	1	string	—	Specifies a script file to run after reconstructing.
log	1	string	—	Specifies a path and file name for logging during the command operation.

Return value

void

Examples

```
// Divide up a long reconstruction
reconstructChunk "Take1.x2d" 0 2005 -fileName "Chunk1.c3d";
reconstructChunk "Take1.x2d" 2000 4005 -fileName "Chunk2.c3d";
reconstructChunk "Take1.x2d" 4000 6005 -fileName "Chunk3.c3d";
reconstructChunk "Take1.x2d" 6000 8005 -fileName "Chunk4.c3d";
reconstructChunk "Take1.x2d" 8000 10000 -fileName "Chunk5.c3d";
reassemble "Chunk1.c3d" "Chunk2.c3d" "Chunk3.c3d" "Chunk4.c3d" "Chunk5.c3d";
```

Additional information

Related commands

- [reassemble \(page 879\)](#)
- [reconstruct \(page 882\)](#)
- [reconstructOptions \(page 887\)](#)

reconstructOptions

Description

Summary

Sets the options for offline reconstruction.

Details

The options match those found on the **Reconstruction** tab of the **Processing** panel. The options can be copied to or from the real time processor using the **-toRTK** and **-fromRTK** options.

Functional area

Data editing

Command syntax

Syntax

```
reconstructOptions [-environmentalDriftTolerance float] [-
minCamsToStartTrajectory integer] [-minCamsToContinueTraj integer] [-
reconMinSeparation float] [-minCentroidRadius float] [-maxCentroidRadius
float] [-minReconstructionRadius float] [-maxReconstructionRadius float]
[-computeRadius boolean] [-computeRays boolean] [-fitMethod string] [-
predMatchFactor float] [-minTrajLength integer] [-pack boolean] [-
startupError float] [-predictionError float] [-minVolume vector] [-
maxVolume vector] [-numThreads integer] [-reset] [-fromRTK] [-toRTK]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
environmentalDriftTolerance	1	float	—	An uncertainty applied (in mm) to camera calibration to tolerate drift in the calibration after calibration due to environmental factors such as temperature change.
minCamsToStartTrajectory	1	integer	—	Specifies the minimum number of cameras needed to start a trajectory. Valid values range from 2 to the number of cameras on your system. The default value is 2.
minCamsToContinueTraj	1	integer	—	Minimum number of cameras needed to continue a trajectory.
reconMinSeparation	1	float	—	The distance (in mm) between the centers of the reconstructions must be greater than this number to be considered valid reconstructions.
minCentroidRadius	1	float	—	Filters out centroids smaller than the minimum radius (in pixels). The default is 0.0 pixels.
maxCentroidRadius	1	float	—	Filters out centroids larger than the maximum radius (in pixels). The default is 50.0 pixels.
minReconstructionRadius	1	float	—	Reconstructions with a radius less than the minimum reconstruction radius (in mm) are discarded by the reconstructor. The default is 0.0 mm.
maxReconstructionRadius	1	float	—	Reconstructions with a radius greater than the maximum reconstruction radius (in mm) are discarded by the reconstructor. The default is 1000.0 mm.

Name	Flag arguments	Argument type	Exclusive to	Comments
computeRadius	1	boolean	—	Specifies if the marker radius should be computed and stored on the radius channel
computeRays	1	boolean	—	Specifies if the ray contributions should be computed and stored.
fitMethod	1	string	—	Specifies the fit method used to create trajectories. Options are "2D Tracks", or "3D Predictions".
predMatchFactor	1	float	—	When using the 3D Predictions Trajectory Fitting method, specifies the degree to which a reconstructed point position should be influenced by its predicted location. Can help with smoothing trajectory and minimizing jitter.
minTrajLength	1	integer	—	Specifies the minimum length a trajectory must have (in frames) to be considered valid.
pack	1	boolean	—	Specifies if trajectories should reuse existing markers when created.
startupError	1	float	—	Specifies the amount of allowable trajectory fitting error (in mm/s) when fitting starts. The default is 150.0 mm/s.
predictionError	1	float	—	Specifies allowable amount of prediction error (in mm/s) when trajectory fitting. The default is 150.0 mm/s.
minVolume	1	vector	—	Specifies one corner of a cube that represents the volume size. Trajectories outside the volume will not be created.

Name	Flag arguments	Argument type	Exclusive to	Comments
maxVolume	1	vector	—	Specifies one corner of a cube that represents the volume size. Trajectories outside the volume will not be created.
numThreads	1	integer	—	Specifies the number of threads used by the reconstructor. To enable auto selection based on CPU core count, set value to 0.
reset	0	—	—	Resets the options to their default values.
fromRTK	0	—	—	Copies the settings on the real time processor to the offline reconstructor.
toRTK	0	—	—	Copies the settings from the offline reconstruction to the real time processor.

Return value

void

Examples

```
// Set the reconstruction volume size to a 10x10 meter space with a
// high of 4 meters assuming a Y up volume.
reconstructOptions -minVolume <<-500, 0, -500>>;
reconstructOptions -maxVolume <<500, 400, 500>>;
```

Additional information

Related commands

- [createReconstructorScript \(page 277\)](#)
- [reconstruct \(page 882\)](#)
- [reconstructChunk \(page 884\)](#)

redo

Description

Summary

Returns your scene file to its state immediately prior to the most recent undo operation.

Details

Shogun Post maintains a history stack of (by default) 20 undoable operations, and will allow you to redo in one step or an optional `[numLevels]` number of those operations.

Note

Shogun Post only records changes (and hence allows undoing) of actual data itself; changes to selection state and various GUI controls are not undoable.

Functional area

System

Command syntax

Syntax

```
redo "numLevels"
```

Arguments

Name	Type	Required	Comments
<code>numLevels</code>	integer	no	One argument is available and it is optional. It specifies the number of undo operations to perform in one step. Not providing this argument will redo the most recent operation (i.e. the same as specifying a value of one.

Flags

None

Return value

void

Examples

```
// Redo the last operation  
redo;
```

Additional information

Related commands

- [undo \(page 1324\)](#)
- [undoConsolidated \(page 1326\)](#)

reduceKeys

Description

Summary

Reduce keys by removing extra keys

Details

Use this command to remove extra keys from a marker where the position of the marker doesn't change between frames. There must exist at least 4 consecutive frames of non-movement before any markers are removed; then the first two and the last frames in the set of unmoving keys are preserved (e.g. if the marker doesn't move between frames 1-10, this command would only keep frames 1,2, and 10.)

Functional area

Data editing

Command syntax

Syntax

```
reduceKeys [-all] [-ranges]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	ranges	Apply the reduction filter to all channels in the module
ranges	0	—	all	Apply the reduction filter only to the keys contained within the selected ranges.

Return value

void

Examples

```
// Select a marker
select Marker1;

// Reduce the keys on this marker
reduceKeys -all;
```

refreshEclipse

Description

Summary

Refreshes the current Eclipse database.

Details

Enables you to refresh the current Eclipse database (scripting equivalent of pressing F5) to display newly imported files.

Functional area

Data management

Command syntax

Syntax

```
refreshEclipse
```

Arguments

None

Flags

None

Return value

void

Examples

```
refreshEclipse;
```

Additional information

Related commands

- | [getEclipseActiveTrial \(page 515\)](#)
- | [getEclipseAssociatedCalibration \(page 517\)](#)
- | [getEclipseAssociatedSubjects \(page 519\)](#)
- | [getEclipseMarkedTrials \(page 521\)](#)
- | [openEclipseDatabase \(page 822\)](#)

remoteControl

Description

Summary

Remote control

Details

This command is valid on machines which are acting as remote servers. remoteControl enables /disables/toggles the forwarding and execution of commands on a connected client.

Commands issued on the server subsequent to the remoteControl command will be executed on both the server and on the client.

Functional area

Remote control

Command syntax

Syntax

```
remoteControl master
```

Arguments

Name	Type	Required	Comments
on/off/toggle	boolean	yes	Enables/disables/toggles remote control

Flags

None

Return value

void

Examples

```
// This block of commands is to be executed on the machine that will
// be receiving commands. A Real Time Processor should be created and
// visible in the client VPL view after the command is sent from the
// server via the remoteControl command forwarding mechanism.
client on;

// this block of commands is to be executed on the machine that will
// be sending commands. A Real Time Processor should be created and
// visible in the server VPL view after the command is sent from the
// server via the remoteControl command forwarding mechanism.
server on;
remoteControl on;
create RealTimeProcessor RealTimeProcessor_1;
```

Additional information

Related commands

- [client \(page 210\)](#)
- [sendRemote \(page 1013\)](#)
- [server \(page 1015\)](#)

removeAllParameters

Description

Summary

Removes all parameters from the selected character.

Details

Removes all parameters from the selected character unless the `-onMod` option is used, in which case removes all parameters from the specified character. Has options for static or dynamic parameters only.

Functional area

Parameters

Command syntax

Syntax

```
removeAllParameters [-onMod string] [-staticOnly][-dynamicOnly] [-solving]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Defines the character to remove all parameters from.
staticOnly				
dynamicOnly				
solving				

Return value

void

Examples

```
// Remove all parameters from the selected character
removeAllParameters;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [setParameter \(page 1118\)](#)

removeBone

Description

Summary

Deletes the selected bone and optionally parents its child to its parent.

Details

Functional area

Skeletal solving

Command syntax

Syntax

```
removeBone [-reparentChild]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
reparentChild				Parents deleted bone's child to its parent.

Return value

void

Additional information

Related commands

- [insertBone \(page 745\)](#)

removeFromClip

Description

Summary

Removes selected module(s) from the active clip.

Details

Removes selected module(s) from the active clip. The animation is also deleted when the object is removed from the active clip. Useful when an object has been inadvertently added or created on a clip that it was not intended for.

Functional area

Data manipulators

Command syntax

Syntax

```
removeFromClip
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Selects all Markers and removes their data from the active clip  
selectByType "Marker";  
removeFromClip;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

removeLayer

Description

Summary

Remove indicated layer from the active clip

Details

Remove indicated layer from the active clip. The error will be generated if there is no clip on the scene.

Functional area

NLE

Command syntax

Syntax

```
removeLayer "layerName"
```

Arguments

Name	Type	Required	Comments
layerName	string	yes	The name of the layer to remove

Flags

None

Return value

void

Examples

```
// Print the name of the active clip
string $clip = `getActiveClip`;
print $clip;
// Add new layer
string $name = "new layer";
addLayer $name;
// Print all layers
string $layers [];
$layers = `getLayers`;
print $layers;

// Remove layer
removeLayer $name;
// Print all layers
$layers = `getLayers`;
print $layers;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

removeMarker

Description

Summary

Remove the marker with the given name from a character.

Details

Removes the marker, all the labeling constraints using it and all parameters associated with it (if no other constraints are using those parameters).

The character that the marker belongs to must either be selected or specified via the `onMod` option.

Functional area

Labeling

Command syntax

Syntax

```
removeMarker markerName[-onMod string]
```

Arguments

Name	Type	Required	Comments
markerName	string	yes	The name of the marker to remove

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod		string		Defines the character from which the marker will be removed

Return value

void

removeMarkerConnection

Description

Summary

Removes the connecting line between markers in the scene.

Details

Removes the connecting line between markers in the scene. If no target markers are specified, all connecting lines originating from the source marker will be removed. Otherwise, only connecting lines between the source and each specified target marker will be removed.

Functional area

Data editing

Command syntax

Syntax

```
removeMarkerConnection "sourceNode" ["targetNode1" "targetNode2"...]
```

Arguments

Name	Type	Required	Comments
targetMarker1	string	no	Name of the marker connected to source.
sourceMarker	string	yes	Name of the marker to remove connection(s) from.

Flags

None

Return value

void

Examples

```
// Draw connecting lines around the four head markers, and then
// remove some of them.
setMarkerConnection "LFHD" "LBHD" -color 0 255 0;
setMarkerConnection "LBHD" "RBHD" -color 0 255 0;
setMarkerConnection "RBHD" "RFHD" -color 0 255 0;
setMarkerConnection "LFHD" "RFHD" -color 0 255 0;

// Now remove the connection from the LBHD to the RBHD
removeMarkerConnection "LBHD" "RBHD";

// And remove all connections originating from the LFHD
removeMarkerConnection "LFHD";
```

Additional information

Related commands

- [getMarkerConnection \(page 594\)](#)
- [setMarkerConnection \(page 1106\)](#)

removeNamespace

Description

Summary

Removes a selected or specified namespace.

Details

All modules under the same parent can be considered in the namespace of that parent, using slashes (/) to separate each module's parent's name. `removeNamespace` operates on all selected modules unless you use the `-onMod` flag to specify the module on which to operate by name.

Note that you may specify the path in commands that require a module name, but you may not specify the path as a module name in the `Name` attribute.

You need to specify only what is required to uniquely identify the name.

Note also that you can remove a single namespace (for example, `removeNamespace "Joe";`) from anywhere in a name containing multiple namespaces (for example, `"Bob:Joe:LUPA"`), to result in `"Bob:LUPA"`.

Functional area

Namespace

Command syntax

Syntax

```
removeNamespace namespace[-onMod string]
```

Arguments

Name	Type	Required	Comments
namespace	string	Yes	Specify the path of a namespace using slashes (/), or colons (:). The number of namespaces in a module name is not limited. Note that if the module contains the same namespace twice, removeNamespace removes only one instance.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies the module on which the command operates.

Return value

void

Example

```
// Remove a namespace called Bob  
removeNamespace "Bob";
```

Additional information

Related commands

- | [addNamespace \(page 117\)](#)
- | [getNamespace \(page 606\)](#)
- | [getNamespaces \(page 608\)](#)
- | [replaceNamespace \(page 927\)](#)

removeParameter

Description

Summary

Removes the given parameter from the selected character.

Details

Removes a parameter with the given name from the selected character or the one specified using `-onMod`.

Functional area

Parameters

Command syntax

Syntax

```
removeParameter parameterName [-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	String	Yes	Defines the name of the parameter to be removed.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Defines the character to remove the parameter from.
solving				

Return value

void

Examples

```
// Remove the parameter named "UpperArmLength" from the selected
// character
removeParameter "UpperArmLength";
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [setParameter \(page 1118\)](#)

removeRayContributions

Description

Summary

To minimize file size, removes camera ray contributions before a .vdf is saved.

Details

By default, this command removes ray contribution information from all markers and cameras in the scene. To remove ray contribution information from only selected markers and cameras, use the command with the `-selectedOnly` flag.

To minimize file size, camera ray contributions are removed before a .vdf is saved.

Functional area

Data Manipulators

Command syntax

Syntax

```
removeRayContributions [-selectedOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>selectedOnly</code>			—	Specifies that camera ray contribution information is removed from selected markers and cameras only

Return value

void

Additional information

Related commands

- [cameraView \(page 196\)](#)
- [getContributingCameras \(page 487\)](#)

removeSoftwareFitCentroids

Description

Summary

Removes cached circle-fitting information.

Details

By default, *.vdf* files are saved with information that enables software-fitted circles to be displayed. This command removes the cached circle-fitting information from the current *.vdf*. Unless the `selectedOnly` flag is used, circle-fitting information is removed for all centroids. It is useful when you want to reconstruct with software circle-fitting information, but want to remove it afterwards to reduce *.vdf* file size.

Functional area

Data manipulators

Command syntax

Syntax

```
removeSoftwareFitCentroids [-selectedOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selectedOnly				Restricts the removal of circle-fitting data to the selection only.

Return value

void

removeUnusedParameters

Description

Summary

Removes unused parameters from the selected character.

Details

Removes unused parameters from the selected character unless `-onMod` option is used to specify a character.

Functional area

Parameters

Command syntax

Syntax

```
removeUnusedParameters [-onMod string] [-solving]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Defines the character to remove unused parameters from.
solving				

Return value

void

Examples

```
// Remove all unused parameters from the selected character  
removeUnusedParameters;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [renameParameter \(page 921\)](#)
- | [setParameter \(page 1118\)](#)

renameParameter

Description

Summary

Renames a parameter.

Details

Renames a parameter. The parameter to be renamed will be looked for on the selected character unless the `-onMod` option is used.

Functional area

Parameters

Command syntax

Syntax

```
renameParameter parameterName newName [-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
<code>parameterName</code>	String	Yes	The name of the parameter to be renamed.
<code>newName</code>	String	Yes	The new name of the parameter.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Defines the character with parameter to rename.
solving				

Return value

void

Examples

```
// Rename the parameter UpperArmLength to LowerArmLength.
renameParameter "UpperArmLength" "LowerArmLength" ;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [setParameter \(page 1118\)](#)

renameShelfGroup

Description

Summary

Changes the name of the shelf group "nameFrom" to "nameTo".

Details

Use renameShelfGroup to rename a shelf group.

Functional area

Interface

Command syntax

Syntax

```
renameShelfGroup "tabName" "nameFrom" "nameTo"
```

Arguments

Name	Type	Required	Comments
tabName	string	Yes	The string name of the tab
nameFrom	string	Yes	The old name of the shelf group to be renamed
nameTo	string	Yes	The new string name of the shelf group

Flags

None

Return value

void

Additional information

Related commands

- [createShelfGroup \(page 285\)](#)
- [deleteShelfGroup \(page 348\)](#)

renameShelfTab

Description

Summary

Changes the name of the `nameFrom` tab to `nameTo`. The tab `nameFrom` must exist on the script shelf, or the command will error.

Details

Use `renameShelfTab` to rename tabs on the shelf.

Functional area

Interface

Command syntax

Syntax

```
renameShelfTab "nameFrom" "nameTo"
```

Arguments

Name	Type	Required	Comments
<code>nameTo</code>	string		The new string name of the shelf tab to be renamed
<code>nameFrom</code>	string		The string name of the shelf tab to be renamed

Flags

None

Return value

void

Examples

```
renameShelfTab "filters" "R_Filters";  
// Changes the name of the shelf tab "filters" to "R_Filters".
```

Additional information

Related commands

- | [createShelfTab \(page 289\)](#)
- | [deleteShelfTab \(page 350\)](#)

replaceNamespace

Description

Summary

Replaces all instances of a selected or specified namespace.

Details

All modules under the same parent can be considered in the namespace of that parent, using slashes (/) to separate each module's parent's name. `replaceNamespace` operates on all selected modules unless you use the `-onMod` flag to specify the module on which to operate by name.

Note that you may specify the path in commands that require a module name, but you may not specify the path as a module name in the Name attribute.

You need to specify only what is required to uniquely identify the name.

Functional area

Namespace

Command syntax

Syntax

```
replaceNamespace oldNamespace newNamespace[-onMod string]
```

Arguments

Name	Type	Required	Comments
oldNamespace	string	Yes	Specify the path of a namespace using slashes (/), or colons (:). The number of namespaces in a module name is not limited. Note that if the module contains the same namespace twice, <code>replaceNamespace</code> removes all instances.

Name	Type	Required	Comments
<code>newNamespace</code>	string	Yes	Specify the path of a namespace using slashes (/), or colons (:). The number of namespaces in a module name is not limited.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>onMod</code>	1	string	—	Specifies the module on which the command operates.

Return value

void

Example

```
// Replace a namespace called Bob with a namespace called Joe
replaceNamespace "Bob" "Joe";
```

Additional information

Related commands

- [addNamespace \(page 117\)](#)
- [getNamespace \(page 606\)](#)
- [getNamespaces \(page 608\)](#)
- [removeNamespace \(page 911\)](#)

replay

Description

Summary

Loops the playback of the last captured .x2d file and allows scrubbing.

Details

Loops the playback and allows scrubbing of the last captured .x2d file, or of the .x2d file specified with the `-file` flag.

Functional area

Playback control

Command syntax

Syntax

```
replay on/off/toggle [-file string] [-scrub]
```

Arguments

Name	Type	Required	Comments
on/off/toggle	boolean	yes	Turn the replay on or off

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
scrub	0	—	—	Plays through and processes the x2d once, after which the processed data is available for scrubbing.

Return value

void

Examples

```
replay on -file "x2d_file_name_here.x2d";
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

resetClip

Description

Summary

Resets the active clip's time attributes to their original state

Details

Resets the active clip to its original state. Useful when a clip has been cropped or time shifted from its original state (when it was created) and needs to be reset.

Functional area

Data manipulators

Command syntax

Syntax

```
resetClip
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Import a clip w/ data on it
// ...
// Offset and crop the clip
string $clips[] = `getModules -type Clip`;
offsetClips -20 $clips[0];
cropClip -leftOnly -curTime;

// Reset the clip to the original state set when
// the importer created it.
resetClip;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)
- | [tileClips \(page 1311\)](#)

restoreGaps

Description

Summary

Restores gaps from a clip

Details

Restores gaps from a named clip and enables you to specify whether to restore all gaps or those in the currently selected range(s) only.

Functional area

Data Editing

Command syntax

Syntax

```
restoreGaps ["targetClip"] [-all] [-ranges]
```

Arguments

Name	Type	Required	Comments
targetClip	string		Name of clip from which to restore gaps

Flags

See above syntax.

Return value

void

Additional information

Related commands

- [fillGaps \(page 389\)](#)
- [findGap \(page 404\)](#)
- [getGaps \(page 547\)](#)

rewind

Description

Summary

Go to the beginning of the play range

Details

Move the current time indicator on the time bar to the beginning of the current playRange.

Functional area

Playback control

Command syntax

Syntax

```
rewind
```

Arguments

None

Flags

None

Return value

void

Examples

```
rewind;  
// Current time indicator is moved to the beginning of the playRange.
```

Additional information

Related commands

- [fastForward \(page 374\)](#)
- [play \(page 832\)](#)

rigidBody

Description

Summary

Calculates for a rigid body object containing positions and rotations data for selected markers over the selected time ranges and places it on the primary selection.

Details

The rigidBody command will help you:

- Intelligently recover lost data that belongs within a marker relationship that may be thought of as "rigid"
- Generate rotational and translation data to define the movement of the rigid body

Rigid bodies provide the ability to convert the independent translations of a group of marker objects to a set of translational and rotational information applied to a single object. A rigid body defines its local coordinate system based upon the selection order of the markers chosen to create it. The main axis (X) is defined by the first and second selected markers, with the first selected being the origin location of the body. The Y-axis pointing direction of the body tries to match the vector running from the first selection to the third selected marker.

Use of the `nonRigidTolerance` option can prevent errant marker behavior from adversely affecting the behavior of the rigid body.

Understanding rigid bodies is important for the effective use of Shogun Post's marker editing capabilities as well as its skeletal solving functionality.

Functional area

Data manipulators

Command syntax

Syntax

```
rigidBody [-all] [-baseFrame integer] [-useCurrentFrame] [-origin string]
[-nonRigidTolerance float] [-selectNonRigidKeys]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	—	Calculate rigid body data for all time instead of only selected time.
baseFrame	1	integer	useCurrentFrame	Use the given integer frame number as the base frame for calculation of the rigidBody.
useCurrentFrame	0	—	baseFrame	Use the current frame for the definition of the rigid body offsets instead of automatically defining the offsets.
origin	1	string	—	String of the marker name to use as the origin of the rigid body.
nonRigidBodyTolerance	1	float	—	The tolerance (measured in millimeters[float]) for qualifying source node keys as valid or invalid during rigid body calculations. Any keys on source nodes that fall outside this tolerance of being "rigid" with respect to the other source nodes is ignored during rigid body calculations.
selectNonRigidKeys	0	—	—	Selects all keys that were ignored on the source nodes when calculating the rigid body.

Return value

void

Examples

```
select LFWT RFWT RBWT LBWT;  
rigidBody -all -useCurrentFrame;  
// This implementation of the rigidBody command creates  
// rotation and translation animation keys across all time  
// for a rigid body object using the current frame to  
// establish the offsets of the indicated markers.  
// The rigid body information generated is applied in  
// this case to the last selected marker: LBWT.
```

Additional information

Related commands

- [fillGaps \(page 389\)](#) -rigid option
- [makeRigid \(page 785\)](#)

rtDropFix

Description

Summary

Fix dropped frames in captured real-time data.

Details

You can use *.mcp* files that were captured in Vicon Shogun Live as a starting point for processing in Vicon Shogun Post. Because the frame rate during real-time capture can be variable, *.mcp* files may contain missing (dropped) frames. *rtDropFix* fills in the data for these missing frames.

This process reconstructs the frames that are missing, trajectory-fits these reconstructions, and then performs a reverse-label pass to better label the filled-in data. You can edit the resulting data as needed using normal Shogun Post operations and editing tools.

This process requires an *.x2d* file that has the same name as the *.mcp* file.

Functional area

Data Editing

Command syntax

Syntax

```
rtDropFix
```

Arguments

None

Flags

None

Return value

void

runScript

Description

Summary

Run a script snippet or script file.

Details

In Shogun Post, you normally have one script execute another script just by providing the script name of one of the script files in your script directories. However, sometimes you want to construct a script "on the fly" by composing a string that contains the script contents. This command allows you to execute the contents of a string. It also allows you to execute a script file which may not exist within your script directories.

Functional area

System

Command syntax

Syntax

```
runScript "script"[-file] [-noFail]
```

Arguments

Name	Type	Required	Comments
script	string	yes	<p>Either a small script snippet (e.g. <code>print "Hello";</code> or a full path to a script file to execute.</p> <p>Note that all file paths in Shogun Post HSL scripting use forward slashes instead of back-slashes.</p>

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	0	—	—	Specifies that the "script" is a file path rather than a script snippet.
noFail	0	—	—	Specifies that this command (causing the script which calls it to fail) if the execution of "script" fails. The command will then return true or false depending on the successful completion of "script".

Return value

boolean

If the `-noFail` flag is specified, returns whether the script successfully ran. Otherwise always returns true.

Examples

```
// Execute a small script which just prints "Hello World" to the
// command log.
string $str = "Hello World";
runScript $str;

// Alternatively run a script that is in your temp folder
// and find out whether it ran successfully.
boolean $wasSuccessful = `runScript "C:/Temp/MyScript.hsl" -file -noFail`;
print $wasSuccessful;
```

Additional information

Related commands

■ [processFile \(page 843\)](#)

saveFile

Description

Summary

Save native file format to the current filename.

Details

Use saveFile to save your work, maintaining the existing filename or by indicating a new filename. If there is no current filename, you are prompted to supply one.

Functional area

File handling

Command syntax

Syntax

```
saveFile ["filename1"][-e string] [-s] [-append string]
```

Arguments

Name	Type	Required	Comments
filename1			Save to the named file

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
e	1	string	—	Export to named format (e.g. "c3d", "bvh") using string argument
s	0	—	—	Export selected items only
append	1	string	—	String name to be appended to the argument filename (e.g. "_edt")

Return value

string array

Examples

```
saveFile "newFileName.hdf" ;
// The above command saves the current scene to new
// file called "newFileName".

string $Path[] = `saveFile -e csm -s -append _vicon RockOn`;
// Saves the currently selected objects to a file called
// "RockOn_vicon.csm"
print $Path;
```

Additional information

Related commands

- [loadFile \(page 779\)](#)
- [newFile \(page 812\)](#)

saveScript

Description

Summary

Save the script currently in the **Script Editor**.

Details

This command saves as a script the current contents of the **Script Editor**. If no path is specified, a file browser, defaulting to the last saved script location, will open, allowing a path to be specified for saving. Where a path is specified, use forward slash (/) path separators.

Functional area

Interface

Command syntax

Syntax

```
saveScript ["scriptPath"]
```

Arguments

Name	Type	Required	Comments
scriptPath	string	yes	Specifies the path to which to save the script

Flags

None

Return value

void

Examples

```
//Saves the contents of the script editor to the specified path  
//Note: forward slash separators are used.  
saveScript "C:/temp";
```

Additional information

Related commands

- [loadScript \(page 782\)](#)

saveWorkspaceFile

Description

Summary

Saves the workspace layout.

Details

The workspace in Shogun Post is fully configurable. The visibility and position of docking windows can be saved to an XML file and then used to restore the workspace layout at a future time.

Functional area

Interface

Command syntax

Syntax

```
saveWorkspaceFile "fileName"
```

Arguments

Name	Type	Required	Comments
filename	string	yes	Path to a text file to write the XML code for the workspace layout

Flags

None

Return value

void

Examples

```
//save a workspace XML file to the root of C drive  
saveWorkspaceFile "C:/layout.xml";
```

Additional information

Related commands

■ [loadWorkspaceString \(page 783\)](#)

scaleCharacter

Description

Summary

Scale an uncalibrated labeling subject to fit labeled marker data.

Details

This command scales an uncalibrated labeling subject to better fit labeled marker data. It is used during the Label ROM operation to scale the subject to the T-pose labels before autolabeling. It should not need to run unless performing portions of ROM labeling in advanced cases.

Functional area

Labeling

Command syntax

Syntax

```
scaleCharacter
```

Arguments

None

Flags

None

Return value

void

sceneView

Description

Summary

Scene view properties

Details

Use the sceneView command to reorganize and show attachments in the Scene view.

Functional area

Interface

Command syntax

Syntax

```
sceneView [-reorganize] [-autoOrganize boolean] [-showAttachments boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
reorganize	0	—	—	—
autoOrganize	1	boolean	—	—
showAttachments	1	boolean	—	—

Return value

void

Examples

```
sceneView -reorganize -showattachments yes;
```

scriptExists

Description

Summary

Determine if a script exists in Shogun Post's internal list of scripts.

Details

This command enables you to query Shogun Post's internal list of scripts to see whether a particular script exists. This may be useful if you are trying to construct a set of smart scripts which call other scripts based on their existence.

Calling a script which does not exist will cause the calling script to fail, which is almost always undesirable.

This command is largely superseded by the [runScript \(page 941\)](#) command.

Functional area

System

Command syntax

Syntax

```
scriptExists "scriptName"
```

Arguments

Name	Type	Required	Comments
scriptName	string	yes	Name of the script to look for. The script name is the portion of the script file's file name excluding the extension (e.g. CutKeysSmart.hsl would have a name of CutKeysSmart).

Flags

None

Return value

boolean

Returns true if a script with the given name exists in Shogun Post's internal list of scripts.

Examples

```
// See if the CutKeysBigRanges script exists in our script directories.  
//If not don't call the script that relies on it.  
if( `scriptExists "CutKeysBigRanges` == true )  
{  
    ScriptThatCallsCutKeysBigRanges;  
}
```

Additional information

Related commands

■ [runScript \(page 941\)](#)

seekToString

Description

Summary

Searches within a file for a string.

Details

This command can be used to search within a text file for a specified string. It returns a Boolean value to indicate whether the search was successful or not.

It can be used for custom File I/O scripts.

This command can only be run on a file that is already open using [fileOpen \(page 387\)](#) in Read mode. If the file has not been opened with readable flags then seekToString will fail.

Functional area

Disk I/O

Command syntax

Syntax

```
seekToString fileID "searchString"
```

Arguments

Name	Type	Required	Comments
searchString	string	yes	String to search for
fileID	int	yes	ID of file previously opened with fileOpen (page 387)

Flags

None

Return value

boolean

Examples

```
//set search string
string $str = "zb";
int $fileID;
boolean $StringExists;

//open file for reading
int $fileID = `fileOpen "C:/myfile.txt" "r"`;

//search for string
$StringExists = `seekToString $fileID $str`;

//print true or false to log
print $StringExists;

//close file
fileClose $fileID;
```

Additional information

Related commands

- [fileClose \(page 385\)](#)
- [fileOpen \(page 387\)](#)
- [isFileReadable \(page 751\)](#)

select

Description

Summary

Select allows you to isolate a module or multiple modules for operations.

Details

Select commands are a staple of CG software. Shogun Post provides a flexible array of select functions to meet any selection requirement. This flexibility is especially powerful when deployed in scripts.

When handling scenes containing multiple characters, selection functions become even more important; `select -priority` enables you to automatically operate on the identically named modules within multiple character hierarchies without having to explicitly declare verbose paths.

Functional area

Selection

Command syntax

Syntax

```
select "name1" "name2" ...[-all] [-a] [-r] [-removePrimary] [-invert] [-t] [-priority] [-priorityOnly]
```

Arguments

Name	Type	Required	Comments
name2	string	no	String name of a second module in the scene
name1	string	no	String name of any module in the scene

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	—	Operates on all modules in the scene.
a	0	—	r	Adds to the existing selection
r	0	—	a	Removes from the existing selection
removePrimary	0	—	a, r	Removes the primary module from the selection.
invert	0	—	a, r, t	Selects all modules not selected and deselects all modules selected.
t	0	—	a, r, invert	Toggles the selection state of the specified modules
priority	0	—	—	Looks under the priority module first for the modules to select.
priorityOnly	0	—	—	Looks under the priority module only for the modules to select.

Return value

void

Examples

```
// module LMT1 will be the sole selection after executing this sequence.
select LHEL LANK LTOE LMT1 LMT5;
select -r LMT1;
select -t LHEL LANK LTOE LMT1 LMT5;
```

Additional information

Related commands

- | [selectabilityOn \(page 959\)](#)
- | [selectByMarkerRadius \(page 963\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectabilityOn

Description

Summary

Sets the selectability to on for all the modules in the scene.

Details

Sets the selectability to on for all the modules in the scene. This command can be used to make objects whose selectability has been turned off selectable again.

Functional area

Selection

Command syntax

Syntax

```
selectabilityOn
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Turn on selectability for all modules in the scene.  
selectabilityOn;
```

Additional information

Related commands

- [hide \(page 741\)](#)
- [select \(page 956\)](#)
- [show \(page 1193\)](#)

selectBranch

Description

Summary

Selects a portion of a hierarchy.

Details

The selectBranch command is used to select the members of a hierarchical branch. The command will continue to select children down or up a branch hierarchy until it reaches the end or beginning of a chain, or a new branching.

Functional area

Selection

Command syntax

Syntax

```
selectBranch node1 node2 ....[-a] [-r] [-u]
```

Arguments

Name	Type	Required	Comments
node(s)	string	yes	String name of node(s) to operate on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Adds to the current selection in the scene

Name	Flag arguments	Argument type	Exclusive to	Comments
r	0	—	a	Removes from the current selection in the scene
u	0	—	—	Selects all modules up the chain until it runs into a new branch

Return value

void

Examples

```
// This command adds all members of the left arm chain below
// and including the lclavicle to the current selection.
// The same applies to the left thigh and below.
selectBranch -a lclavicle lfemur;
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectByMarkerRadius \(page 963\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectByMarkerRadius

Description

Summary

Selects markers by marker radius.

Details

Markers have a radius channel. When reconstructing in Shogun Post, it's possible for reconstruction to place the radius of the marker at each frame on the radius channel. The `selectByMarkerRadius` command allows markers with an average radius between the min and max radius arguments to be selected. This can be very useful for selecting all face or finger markers separately from body makers, even when the markers are still unlabeled.

Functional area

Selection

Command syntax

Syntax

```
selectByMarkerRadius minRadius maxRadius[-a] [-r] [-childOf string]
```

Arguments

Name	Type	Required	Comments
minRadius	float	yes	Markers with a radius smaller then this value will not be selected.
maxRadius	float	yes	Markers with a radius larger then this value will not be selected.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Adds to the current selection.
r	0	—	a	Removes from the current selection.
childOf	1	string	—	Selects only markers that are a child of the given module.

Return value

void

Examples

```
// Select all markers with a radius between 2 and 6 under the
// character named Bob.
selectByMarkerRadius 2.0 6.0 -childOf "Bob";
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectByName

Description

Summary

Selects all modules with matching name(s).

Details

Selects all modules in the scene with matching name(s).

Names can contain DOS-style wild card characters to select modules matching name patterns. The wild card character ? matches any single character, while * matches zero or more characters.

Thus, the following command selects all markers whose name began with L (usually indicating they are on the left of the body):

```
selectByName "L*" -type "Marker";
```

Unlike the [select \(page 956\)](#) command which selects only the first module with the given name if multiple modules of the same name exist, the selectByName command selects all modules of the given name.

Functional area

Selection

Command syntax

Syntax

```
selectByName "name1" "name2" ...[-a] [-r] [-type string] [-childOf string]
```

Arguments

Name	Type	Required	Comments
name1	string	no	One or more module names to select. The name can contain DOS-style wild card characters to match name patterns. If no names are specified, all modules are de-selected.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Specifies that selection operation add to the current scene selection. The default is to replace the scene selection.
r	0	—	a	Specifies that selection operation be removed from the current scene selection. The default is to replace the scene selection.
type	1	string	—	Narrows the search to specific module types (e.g. Bone, Marker, etc.). The default is all module types.
childOf	1	string	—	Only selects modules that are children of this module name.

Return value

void

Examples

```
// Select all head markers in the scene, not just the
// one under the priority search module
selectByName "LFHD" "RFHD" "LBHD" "RBHD";

// Could have also done this
selectByName "*HD";
```

```
// Add to the selection all front waist markers  
selectByName "?FWT" -a;
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectByParameter

Description

Summary

Selects nodes using the given parameter.

Details

Parameters can be used on bones and constraints. The selectByParameter command can be used to select the bones and constraints that use the given parameter.

Functional area

Parameters

Command syntax

Syntax

```
selectByParameter parameterName [-onMod string] [-a] [-r] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	Yes	The name of the required parameter

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Specifies the character that owns the parameter.
a	0	—	r	Adds to the current selection.
r	0	—	a	Removes from the current selection.
solving				

Return value

void

Examples

```
// Select all the nodes that use the parameter named LowerLegLength
// on character Bob
selectByParameter "LowerLegLength" -onMod -Bob;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [getParameter \(page 627\)](#)
- | [getParameters \(page 633\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [setParameter \(page 1118\)](#)

selectByPart

Description

Summary

Selects bone modules in the scene by their Part_Name attribute.

Details

Selects all bone modules in the scene that have the given part name(s). The Part_Name attribute on bone modules is used for retargeting operations, which relies on body part naming.

Functional area

Selection

Command syntax

Syntax

```
selectByPart partName1 partName2...[-a] [-r] [-childOf string] [-side string]
```

Arguments

Name	Type	Required	Comments
partName(s)	string	yes	The part name(s) to select by. Must be one of the names listed in the bone module Part_Name attribute.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Specifies that selection operation add to the current scene selection. The default is to replace the scene selection.
r	0	—	a	Specifies that selection remove from the current scene selection. The default is to replace the scene selection.
childOf	1	string	—	Only selects modules that are children of this module name.
side	1	string	—	Only selects modules that have the side attribute set to this.

Return value

void

Examples

```
// Select all bone modules with the "Collar" part name.
selectByPart "Collar";
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByName \(page 965\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)

-
- [selectSet \(page 1003\)](#)
 - [selectTree \(page 1011\)](#)
 - [setPrimary \(page 1126\)](#)

selectBySide

Description

Summary

Selects bone modules in the scene by their Side attribute.

Details

Selects all bone modules in the scene that have the given side(s). The Side attribute on bone modules is used for retargeting operations, which relies on body part naming.

Functional area

Selection

Command syntax

Syntax

```
selectBySide sideName1 sideName2...[-a] [-r] [-childOf string]
```

Arguments

Name	Type	Required	Comments
sideName (s)	string	yes	The side(s) to select by. Must be one of the sides listed in the bone module Side attribute.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Specifies that selection operation add to the current scene selection. The default is to replace the scene selection.
r	0	—	a	Specifies that selection operation remove from the current scene selection. The default is to replace the scene selection.
childOf	1	string	—	Only selects modules that are children of this module name.

Return value

void

Examples

```
// Select all bone modules with the "Right" side attribute.
selectBySide "Right";
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectByType

Description

Summary

This command allows you to select all the objects of a named type or types in your scene. Using no options deselects all other objects first, then selects only the named types.

Details

This command allows you to identify and select all the objects in the scene of a selected type. selectByType provides the ability to do things like:

- Select all solvers to activate or deactivate them, or
- Select all constraints to reset them, or
- Select all bones to cutKeys and clear the animation from them.

Functional area

Selection

Command syntax

Syntax

```
selectByType [-a] [-r] [-childOf string]
```

Arguments

Name	Type	Required	Comments
Type	string	yes	The type of module to select.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	—
r	0	—	a	—
childOf	1	string		Only selects modules that are children of this module name.

Return value

void

Examples

```
// These commands select all the Bone modules in your
// scene, and then remove all the animation from those modules
// resetting the skeleton(s) to the base pose.
selectByType Bone;
cutKeys -all;
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectChildren

Description

Summary

Selects the children of each specified module.

Details

Use selectChildren when you need to add or subtract the children of the specified modules. You can also select all descendants of the specified module using the recursive option.

Functional area

Selection

Command syntax

Syntax

```
selectChildren node1 node2 ....[-a] [-r] [-recursive]
```

Arguments

Name	Type	Required	Comments
node(s)	string	yes	String name of node(s) to operate on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Adds to the current selection in the scene

Name	Flag arguments	Argument type	Exclusive to	Comments
r	0	—	a	Removes from the current selection in the scene
recursive	0	—	—	Selects all children of the nodes specified

Return value

void

Examples

```
// This command selects the indicated nodes, and each
// of their immediate children.
selectChildren lcollar rcollar;
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectClipObjects

Description

Summary

Selects the modules that exist on the specified clip(s).

Details

Selects the modules that exist on the specified clip(s). More than one clip can be specified. Can be used in conjunction with the [removeFromClip \(page 903\)](#) command.

Functional area

NLE

Command syntax

Syntax

```
selectClipObjects clip ...[-a] [-r]
```

Arguments

Name	Type	Required	Comments
clip	string	yes	The clip whose modules you want selected. May specify multiple clips by name or specify an array of clips. If no clips are specified, then modules with data for the active clip will be selected.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Add modules to current selection
r	0	—	a	Remove modules from current selection

Return value

void

Examples

```
// Select all modules with data for the active clip, adding to the
// scene selection
selectClipObjects -a;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [setActiveClip \(page 1017\)](#)

-
- [setActiveClip \(page 1017\)](#)
 - [setActiveLayer \(page 1020\)](#)
 - [tileClips \(page 1311\)](#)

selectDropListItem

Description

Summary

Select a user drop list item.

Details

Selects an item in the drop list user control specified by `userControlID`. The selected item is the item that is displayed when the drop list is collapsed. When it is dropped, the selected item is highlighted. If no item is selected, no text is displayed when the drop list is collapsed, and no items are highlighted when dropped.

To remove selection, pass a -1 as the index argument.

Functional area

User Window

Command syntax

Syntax

```
selectDropListItem userControlID indexOrString
```

Arguments

Name	Type	Required	Comments
index	int	yes	Zero based index of the item to select.
userControlId	int	yes	ID of user control to operate on.

Flags

None

Return value

boolean

Examples

```
// Select an item in a Drop List User Control.
int $windowId;
int $controlId;
int $index;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createDropList $windowId`;

// Add some items to the Drop List.
addDropListItem $controlId "Item 1";
$index = `addDropListItem $controlId "Item 2"`;
addDropListItem $controlId "Item 3";

// Select the item whose index we just got.
selectDropListItem $controlId $index;
```

Additional information

Related commands

- | [addDropListItem \(page 105\)](#)
- | [createDropList \(page 247\)](#)
- | [deleteAllDropListItems \(page 324\)](#)
- | [deleteDropListItem \(page 336\)](#)
- | [findDropListItem \(page 401\)](#)
- | [getDropListItem \(page 510\)](#)
- | [getDropListSelectedItem \(page 513\)](#)
- | [getNumDropListItems \(page 612\)](#)
- | [setDropListHandler \(page 1058\)](#)

selectionSet

Description

Summary

Create a selection set with optional name or operate on the named selection set.

Details

Use this command when you need to create, delete, add nodes or sets to, or remove nodes or sets from selection sets.

Functional area

Selection

Command syntax

Syntax

```
selectionSet "setName" ["nodeName1"] ...[-add] [-set string] [-remove] [-delete] [-deleteAll]
```

Arguments

Name	Type	Required	Comments
setName	string	yes	The name of a selection set to be operated on, if none with indicated name exists, a selection set having that name is created.
nodeName (s)	string	no	The name of node(s) to be added or removed from the set.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
add	0	—	remove, delete, deleteAll	Adds the currently selected markers, and other sets (using -set) to "name" set
set	1	string	—	Specifies a selection set to be added, or removed from the given selection set depending if -add or -remove are used.
remove	0	—	add, delete, deleteAll	Removes selected markers in scene and other sets (using -set) from "name" set
delete	0	—	remove, deleteAll	Deletes indicated set
deleteAll	0	—	remove, delete	Deletes all selection sets. Use with care.

Return value

void

Examples

```
// Create a new selection set called "leftWaist" and
// add the markers LMWT, LFWT, and LBWT to it.
select LMWT LFWT LBWT;
selectionSet leftWaist;
```

Additional information

Related commands

- [getSelectionSets \(page 956\)](#)
- [getSelectionSetSets \(page 682\)](#)
- [isSelectionSet \(page 757\)](#)

-
- [select \(page 956\)](#)
 - [selectSet \(page 1003\)](#)

selectKeys

Description

Summary

Selects or deselects keys on the selected channels.

Details

Use `selectKeys` when you need to work with specific key sets; Shogun Post allows single or multiple range selections. To select keys only within certain properties: Translation, or Rotation for example, use [selectProperty \(page 996\)](#) to include/exclude them.

Functional area

Selection

Command syntax

Syntax

```
selectKeys [-a] [-r] [-invert] [-all] [-ranges] [-primaryOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r, invert	Add keys in selected time ranges to current key selection
r	0	—	a, invert	Remove keys in selected time ranges from current key selection

Name	Flag arguments	Argument type	Exclusive to	Comments
invert	0	—	a, r, all	Inverts the selection from the currently selected keys to the unselected keys
all	0	—	ranges	Select all keys of all selected properties.
ranges	0	—	all	Select all keys on currently active properties that lie within the selected time ranges.
primaryOnly	0	—	—	Selects keys on the primary Module only.

Return value

void

Examples

```
// In this example 3 objects are selected, then a range
// within the playRange, then a subset of the object's properties,
// finally selecting the range of keys falling within the
// indicated range on the indicated properties.
select LMWT LFWT LBWT;
selectRange 200 240;
selectProperty Translation;
selectKeys -ranges;
```

Additional information

Related commands

- [select \(page 956\)](#)
- [selectRange \(page 998\)](#)

selectListBoxItem

Description

Summary

Select a User List Box item.

Details

Selects an item in the list box user control specified by `userControlId`. Selected items appear highlighted in the list.

If the `-multi` flag was specified with the [createListBox \(page 259\)](#) command, this command will add to the current list box selection. Otherwise, the command will replace the current selection.

To remove selection, pass a `-1` as the index argument.

Functional area

User Window

Command syntax

Syntax

```
selectListBoxItem userControlID index
```

Arguments

Name	Type	Required	Comments
index	int	yes	Zero based index of the item to select.
userControlId	int	yes	ID of user control to operate on.

Flags

None

Return value

integer

Examples

```
// Select an item in a List Box User Control.
int $windowId;
int $controlId;
int $index;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createListBox $windowId`;

// Add some items to the List Box.
addListBoxItem $controlId "Item 1";
$index = `addListBoxItem $controlId "Item 2"`;
addListBoxItem $controlId "Item 3";

// Select the item whose index we just got.
selectListBoxItem $controlId $index;
```

Additional information

Related commands

- | [addListBoxItem \(page 110\)](#)
- | [createListBox \(page 259\)](#)
- | [deleteAllListBoxItems \(page 326\)](#)
- | [deleteListBoxItem \(page 340\)](#)
- | [findListBoxItem \(page 407\)](#)
- | [getListBoxItem \(page 578\)](#)
- | [getListBoxSellItems \(page 581\)](#)
- | [getNumListBoxItems \(page 616\)](#)
- | [setListBoxHandler \(page 1088\)](#)

selectListViewItem

Description

Summary

Set the selection state of an item in a list view.

Details

This command allows you to set the selection state of an item in a list view.

Functional area

User Window

Command syntax

Syntax

```
selectListViewItem userControlID row select
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the list view to operate on
row	integer	yes	The row to be selected
select	boolean	yes	The new selected state of the row.

Flags

None

Return value

void

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columns
string $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the first
// column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
```



```
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";
// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob`s name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [deleteListViewItem \(page 342\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

selectParent

Description

Summary

Selects the immediate parent of each named node

Details

Use selectParent when you need to add or remove the parents of the nodes specified to/from your current selection.

Functional area

Selection

Command syntax

Syntax

```
selectParent node1 node2 ...[-a] [-r] [-recursive]
```

Arguments

Name	Type	Required	Comments
node1	string	yes	Name of module in the scene to operate on
node2	string	no	Additional modules can specified

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Adds to the current selection in the scene
r	0	—	a	Removes from the current selection in the scene
recursive	0	—	—	Selects all parents of the nodes indicated.

Return value

void

Examples

```
// This command will select the indicated nodes and their
// immediate parents.
selectParent -r lcollar rcollar;
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectSet \(page 1003\)](#)
- | [selectTree \(page 1011\)](#)
- | [setPrimary \(page 1126\)](#)

selectProperty

Description

Summary

Select the named properties. No arguments deselects all properties.

Details

Use this command for flexible control over the which channels operations will apply to. You can choose to select individual channels (like TranslationX) or can all translation or rotation channels.

Note that the Graph view only displays selected channels.

Functional area

Selection

Command syntax

Syntax

```
selectProperty "name1" "name2" ...[-a] [-r] [-t]
```

Arguments

Name	Type	Required	Comments
name	string	no	Name(s) of properties to select

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Add to currently selected properties
r	0	—	a	Remove from currently selected properties
t	0	—	—	Toggles the currently selected properties

Return value

void

Examples

```
// These commands select the 3 named markers, then a range within
// the playRange, then the X and Z Translation channels of the objects.
select LMWT LFWT LBWT;
selectRange 150 280;
selectProperty "TranslationX" "TranslationZ";

// Adds to selection all three Rotation channels
selectProperty -a "RotationX" "RotationY" "RotationZ";
```

Additional information

Related commands

- [selectKeys \(page 987\)](#)
- [selectRange \(page 998\)](#)
- [setProperty \(page 1130\)](#)

selectRange

Description

Summary

Select a subset, or several subsets, of the current playRange. No arguments deselects all time.

Details

Use the selectRange commands to control your time range selections with a high degree of flexibility.

The `-keys` option may be used to make range selections based upon currently selected keys.

The `-setIn` and `-setOut` commands allow you to work in the Perspective view and make range selections without having to open the Graph view, or declare frames with numeric arguments.

Functional area

Selection

Command syntax

Syntax

```
selectRange startTime endTime[-all] [-a] [-r] [-invert] [-keys] [-setIn]
[-setOut]
```

Arguments

Name	Type	Required	Comments
endTime	integer	no	The last frame of the range to select.
beginTime	integer	no	The first frame of the range to select.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	a, r, invert, keys, setIn, setOut	Select all time
a	0	—	all, r, invert, setIn, setOut	Add to current time selection
r	0	—	all, a, invert, setIn, setOut	Remove from current time selection
invert	0	—	all, a, r, keys, setIn, setOut	Invert current time selection
keys	0	—	all, invert, setIn, setOut	Select time ranges of all currently selected keys on selected modules.
setIn	0	—	all, a, r, invert, keys, setOut	Sets the current frame as one end of a range. Should be followed by a -setOut
setOut	0	—	all, a, r, invert, keys, setIn	Sets the current frame as the other end of a range. Follows a -setIn.

Return value

void

Examples

```
// Select a range of the playRange beginning at frame
// 240 and ending at 285;
selectRange 240 285;
```

Additional information

Related commands

- [select \(page 956\)](#)
- [selectKeys \(page 987\)](#)
- [selectProperty \(page 996\)](#)

selectRelatedKeys

Description

Summary

Selects keys on related channels

Details

Because channels in Shogun Post can be independently selected and operated on, a key that is selected on a channel at a given frame might not be selected on its "related" channels. For example, [findBadData \(page 395\)](#) may select keys on the TranslationX channel at a given frame, but not on the TranslationY or TranslationZ channels.

This command selects the keys at that frame on TranslationY and TranslationZ (if the keys exist).

Functional area

Selection

Command syntax

Syntax

```
selectRelatedKeys
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Find noise on the LFWT marker and select any related
// keys on the noisy channels.
select LFWT;
findBadData 5 0.1 35 -all -selected;

// findBadData might have selected keys "unevenly". Make
// sure keys on "related" channels are selected as well.
selectRelatedKeys;
```

Additional information

Related commands

- [findSelectedKeys \(page 414\)](#)
- [selectKeys \(page 987\)](#)

selectSet

Description

Summary

This command selects objects in the scene via their membership in any selection sets previously defined.

Details

Selection commands are a staple of CG software. Shogun Post provides a flexible array of select functions, including this selectSet command, to meet just about any selection requirement.

The selectSet command coupled with pre-defined selection sets allows you to set up your scenes without regard to specific marker names. For example, a selection set can be defined called setWaist that includes all the names you might encounter for waist markers, allowing the waist markers to be selected via the selection set rather than explicit marker names.

Functional area

Selection

Command syntax

Syntax

```
selectSet "set1" "set2" ...[-all] [-a] [-r] [-priorityOnly]
```

Arguments

Name	Type	Required	Comments
set1	string	no	String names of pre-defined Selection Sets. More than one can be used.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	—	Selects all members of all defined Selection Sets. Using this option removes the need to supply an argument.
a	0	—	r	Add to the current selection
r	0	—	a	Remove from the current selection
priorityOnly	0	—	—	Selects only modules3 that are children of the priority module.

Return value

void

Examples

```
// First add several Selection Sets to the current selection.
// Then remove one.
// Then select all Selection Set members.
selectSet -a head LeftLeg RightFoot;
selectSet -r LeftLeg;
selectSet -all;
```

Additional information

Related commands

- | [getSelectionSets \(page 680\)](#)
- | [getSelectionSetSets \(page 682\)](#)
- | [isSelectionSet \(page 757\)](#)
- | [select \(page 956\)](#)

selectShelfTab

Description

Summary

This command selects a shelf tab.

Details

selectShelfTab can be used to select the different tabs on the Shogun Post ribbon: **Subject Setup**, **Processing**, **Objects**, **Panels**, or any other tab you have created. You can select by name, number, or go through the tabs by Next or Previous.

Functional area

Interface

Command syntax

Syntax

```
selectShelfTab ["tabName" | tabNumber ] [-next] [-prev]
```

Arguments

Name	Type	Required	Comments
tabNumber	int	yes	identifies the tab to select by number
tabName	string	yes	identifies the tab to select by name

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
next	0	—	prev	—
prev	0	—	next	—

Return value

void

Examples

```
// The following example selects the Shelf Tab "Panels":
selectShelfTab "Panels";
```

selectTabItem

Description

Summary

Activate the specified tab.

Details

Activate the specified tab specifically by index, starting with 0.

Functional area

User Window

Command syntax

Syntax

```
selectTabItem userControlID index
```

Arguments

Name	Type	Required	Comments
index	int	yes	The index of the tab you want to make active
userControlId	int	yes	ID of User Window to place the control on.

Flags

None

Return value

integer

Examples

```
// Create a tab and add an additional tab to the main tab.  
int $windowId;  
int $mainTab;  
  
// First create a User Window to place the control on  
$windowId = `createWindow "MyWindow"`;  
  
// create a main tab controller then add a second and third tab  
$mainTab = `createTab $windowId "Main" "Second" "Third"`;  
  
// make the third tab active  
selectTabItem $mainTab 2;
```

Additional information

Related commands

- | [addTab \(page 127\)](#)
- | [createTab \(page 300\)](#)
- | [deleteAllTabItems \(page 330\)](#)
- | [deleteTabItem \(page 352\)](#)
- | [findTabItem \(page 416\)](#)
- | [getTabItem \(page 697\)](#)
- | [getTabSelectedItem \(page 699\)](#)
- | [setTabHandler \(page 1167\)](#)

selectTails

Description

Summary

Select a number of keys on the edges of gaps.

Details

The selectTails command will select a fixed number of keys surrounding a gap.

You may select values representing the number of keys to be selected independently for leading and trailing portions.

If only one value is chosen at execution of the command, that value is considered to be a leading keys value, no trailing keys will be selected.

Functional area

Data manipulators

Command syntax

Syntax

```
selectTails leadingTrim trailingTrim
```

Arguments

Name	Type	Required	Comments
leadingTrim	int	Yes	Lead keys: indicates the number of keys to be selected at the leading edge of a gap
trailingTrim	int	Yes	Trailing keys: indicates the number of keys to be selected following the gap

Flags

None

Return value

void

Examples

```
select LFWT LMWT LBWT;  
findGap;  
selectTails 2 5;  
// This sequence moves the time bar cursor to the next gap it finds  
// in the timeline among the selected markers, LFWT, LMWT, and LBWT.  
// The selectTails command selects 2 frames before the gap, and  
// 5 frames following the gap.
```

Additional information

Related commands

- [findGap \(page 404\)](#)
- [findSelectedKeys \(page 414\)](#)
- [findTail \(page 418\)](#)

selectTree

Description

Summary

Selects the entire tree that each specified module belongs to.

Details

Use selectTree to add or subtract from your selections based upon hierarchical relationships in the scene. You may pass individual nodes, or arrays of nodes to the selectTree command.

Functional area

Selection

Command syntax

Syntax

```
selectTree node1 node2 ...[-a] [-r]
```

Arguments

Name	Type	Required	Comments
node(s)	string	yes	Name of node(s) in the scene to operate on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Adds to the current selection in the scene
r	0	—	a	Removes from the current selection in the scene

Return value

void

Examples

```
// This command adds to the current selection all the
// members of the hierarchy to which the node lcollar belongs.
selectTree -a lcollar;
```

Additional information

Related commands

- | [select \(page 956\)](#)
- | [selectabilityOn \(page 959\)](#)
- | [selectBranch \(page 961\)](#)
- | [selectByName \(page 965\)](#)
- | [selectByPart \(page 970\)](#)
- | [selectBySide \(page 973\)](#)
- | [selectByType \(page 975\)](#)
- | [selectChildren \(page 977\)](#)
- | [selectParent \(page 994\)](#)
- | [selectSet \(page 1003\)](#)
- | [setPrimary \(page 1126\)](#)

sendRemote

Description

Summary

Sends a command to remote control

Details

This command is valid on machines which are acting as remote servers.

sendRemote issues a command to a client connected to a server.

Functional area

Remote control

Command syntax

Syntax

```
sendRemote commandString
```

Arguments

Name	Type	Required	Comments
commandString	string	yes	Specifies the command to send. <code>commandString</code> should end with a semicolon.

Flags

None

Return value

void

Examples

```
// This block of commands is to be executed on the machine which will
// be receiving commands. A Real Time Processor should be created and
// visible in the client VPL view after the command is sent from the
// server via the sendRemote commandclient on;
// this block of commands is to be executed on the machine which will
// be sending commands.
server on;
string $command;
$command = "create RealTimeProcessor RealTimeProcessor_1";
sendRemote $command;
```

Additional information

Related commands

- | [client \(page 210\)](#)
- | [server \(page 1015\)](#)
- | [remoteControl \(page 897\)](#)

server

Description

Summary

Turns the server on and off

Details

This command allows you receive remote script commands.

Functional area

Remote control

Command syntax

Syntax

```
server on/off/toggle [-ip string] [-port string] [-connectPort string]
```

Arguments

Name	Type	Required	Comments
on/off/toggle	boolean	yes	Turn the server on or off

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ip		string	—	Specifies the IP address
port	1	string	—	Specifies the port to use for the network socket connection
connectPort		string		

Return value

void

Examples

```
// This command will enable the server
server on -port 4344;
```

Additional information

Related commands

- [client \(page 210\)](#)
- [remoteControl \(page 897\)](#)
- [sendRemote \(page 1013\)](#)

setActiveClip

Description

Summary

Set the specified clip to be active.

Details

Set the specified clip to be active or specify a clip to be referenced with a `-next` or `-prev` flag. Useful when jumping back and forth between clips in a script.

Functional area

Data manipulators

Command syntax

Syntax

```
setActiveClip "clip" [-next] [-prev]
```

Arguments

Name	Type	Required	Comments
clip	string	no	The clip in which you want to make active or have the command reference. Not required if you use the <code>-next</code> or <code>-prev</code> flags, though if one of those flags is specified, then the clip made active will be relative to this clip.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
next	0	—	prev	Activates the clip before the currently active clip. If the currently active clip is the first clip in the clip list, then it will "wrap" around, selecting the last clip in the clip list.
prev	0	—	next	Activates the clip after the currently active clip. If the currently active clip is the last clip in the clip list, then it will "wrap" around, selecting the first clip in the clip list.

Return value

void

Examples

```
// Create 3 Clips
create Clip "Clip_A" "Clip_B" "Clip_C";
```

```
// Set Clip_B active
setActiveClip "Clip_B";
```

```
// Set Clip_A active
setActiveClip -prev;
```

```
//set Clip_C active
setActiveClip "Clip_B" -next;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)

-
- | [flatten \(page 426\)](#)
 - | [getActiveLayer \(page 437\)](#)
 - | [getClips \(page 471\)](#)
 - | [moveClip \(page 803\)](#)
 - | [offsetClips \(page 820\)](#)
 - | [removeFromClip \(page 903\)](#)
 - | [removeLayer \(page 905\)](#)
 - | [resetClip \(page 931\)](#)
 - | [selectClipObjects \(page 979\)](#)
 - | [setActiveLayer \(page 1020\)](#)
 - | [tileClips \(page 1311\)](#)

setActiveLayer

Description

Summary

Set the active layer in the active clip

Details

Set the active layer in the active clip using the name of the layer or a relative position. The error will be generated if there is no clip on the scene.

Functional area

NLE

Command syntax

Syntax

```
setActiveLayer "layerName" [-next] [-prev]
```

Arguments

Name	Type	Required	Comments
layerName	string	yes	The name of layer to make active. Can be skipped if one of the flags -next or -prev is set.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
next	0	—	prev	Set the next layer in the active clip as an active
prev	0	—	next	Set the previous layer in the active clip as an active

Return value

void

Examples

```
// Print the name of the active clip
string $clip = `getActiveClip`;
print $clip;
```

```
// Add new layers
string $name = "new layer";
addLayer $name;
addLayer $name;
addLayer $name;
setActiveLayer $name;
setActiveLayer -next;
setActiveLayer -prev;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)

-
- [getClips \(page 471\)](#)
 - [moveClip \(page 803\)](#)
 - [offsetClips \(page 820\)](#)
 - [removeFromClip \(page 903\)](#)
 - [removeLayer \(page 905\)](#)
 - [resetClip \(page 931\)](#)
 - [selectClipObjects \(page 979\)](#)
 - [tileClips \(page 1311\)](#)

setActiveTake

Description

Summary

Sets the scene's active take

Details

Sets the scene's active take. The active take is the source capture file (x2d file). There can only be one active take. When you reconstruct, you are reconstructing data from the active take.

Note that calling this method is the equivalent of importing an x2d file. It is provided for convenience and to allow you to "clear" the active take by passing an empty string ("") as the active take.

Functional area

System

Command syntax

Syntax

```
setActiveTake x2dFile[-loadCal] [-loadChars] [-loadHardwareSettings] [-loadVideo] [-moveVideo]
```

Arguments

Name	Type	Required	Comments
x2dFile	string	yes	Full path to the x2d file. If an empty string is passed in, then the scene's active take will be cleared.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
loadCal	0	—	—	Loads the associated calibration file when loading the x2d file
loadChars	0	—	—	Loads the associated characters when loading the x2d file
loadHardwareSettings				
loadVideo				
moveVideo				

Return value

void

Examples

```
// This will set the active take, loading in the calibration file
// associated with the take
setActiveTake "C:/MyProduct/CaptureDay1/Session1/Take0002.x2d" -loadCal;
```

Additional information

Related commands

■ [getActiveTake \(page 441\)](#)

setAltToSelect

Description

Summary

Set whether the ALT key must be pressed when selecting modules in the Shogun Post GUI.

Details

There is variance in the behavior of 3D packages in that some require you to hold down the ALT key when selecting objects in the GUI. The packages that don't require the ALT key usually interpret mouse clicks and drags as pan and orbit operations (depending on the type of view).

Shogun Post allows you to choose which behavior you prefer. By default, Shogun Post uses the ALT-to-select behavior.

Functional area

System

Command syntax

Syntax

```
setAltToSelect altToSelect
```

Arguments

Name	Type	Required	Comments
altToSelect	boolean	yes	Whether the ALT key must be pressed when selecting objects/modules in the Shogun Post GUI.

Flags

None

Return value

void

Examples

```
// Emulate the behavior of Vicon iQsetAltToSelect true;
```

setAutoSaveFile

Description

Summary

Specify the name of the file in which the current file will be auto saved.

Details

Shogun Post has an auto scene saving feature which will automatically save your scene after a specified period of time since the last file save performed.

By default, the location for the file is a file called *AutoSave.hdf* in your Shogun directory (e.g. *C:/Users/Public/Documents/Vicon/ShogunPost#.#/AutoSave.hdf*) unless you change it using the `setAutoSaveFile` command.

Use forward slashes instead of back-slashes for all file system paths.

Functional area

System

Command syntax

Syntax

```
setAutoSaveFile "autoSaveFile"
```

Arguments

Name	Type	Required	Comments
autoSaveFile	string	yes	Name of the file to save when Shogun Post auto-saves the scene

Flags

None

Return value

void

Examples

```
// Sets the auto-save file to be located in C:\Temp\ShogunPostAutoSave.hdf  
setAutoSaveFile "C:/Temp/ShogunPostAutoSave.hdf";
```

Additional information

Related commands

- | [getAutoSaveFile \(page 457\)](#)
- | [setHotKeyFile \(page 1079\)](#)
- | [setLogFile \(page 1104\)](#)
- | [setMarkingMenuFile \(page 1109\)](#)

setBoneLength

Description

Summary

Sets the length of the primary selected BoneNode.

Details

Sets the length of the primary selected BoneNode by adjusting the selected BoneNode's child (in the axis pointing down the bone). Used often when fine-tuning the setup of a skeleton.

Functional area

Data manipulators

Command syntax

Syntax

```
setBoneLength length [-relative] [-adjustChildren]
```

Arguments

Name	Type	Required	Comments
length			Constrains the children of the bone to their current location in world space.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
relative	0	—	—	—
adjustChildren	0	—	—	—

Return value

void

Examples

```
//Demonstrate the setBoneLength command
//Create a default skeleton
select lfemur;
setBoneLength 1.2 -relative -adjustChildren ;
```

Additional information

Related commands

- | [createSkelScript \(page 291\)](#)
- | [solve \(page 1226\)](#)

setButtonShelfFile

Description

Summary

Set the file where Shogun Post loads and saves your button shelves.

Details

Set the file where Shogun Post loads and saves your button shelves.

Functional area

Interface

Command syntax

Syntax

```
setButtonShelfFile ButtonShelfFile [-save]
```

Arguments

Name	Type	Required	Comments
ButtonShelfFile	string	yes	Name of the file where Shogun Post loads and saves your button shelves.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
save	0	—	—	Causes the current button shelf to be saved to the given file. If not specified, then the button shelves in the file will be loaded.

Return value

void

Examples

```
// Save your button shelves to the root of your C: drive
setButtonShelfFile "C:/MyButtonShelf.hsl" -save;
```

Additional information

Related commands

- [setAutoSaveFile \(page 1027\)](#)
- [setLogFile \(page 1104\)](#)
- [setMarkingMenuFile \(page 1109\)](#)

setCheckBoxCheck

Description

Summary

Set the user check box check value.

Details

Sets the check value of the check box user control specified by `userControlId`. Because check boxes can be created with the `-triState` flag (the third state being "indeterminate"), either a Boolean or an integer is accepted for `checkVal`.

Functional area

User Window

Command syntax

Syntax

```
setCheckBoxCheck userControlID check
```

Arguments

Name	Type	Required	Comments
<code>checkVal</code>	int	yes	0 to uncheck the control, 1 to check it, or 2 to set it to indeterminate
<code>userControlId</code>	int	yes	ID of User Control to operate on.

Flags

None

Return value

void

Examples

```
// Set the checked value of a Check Box User Control
int $windowId;
int $controlId;
int $checked;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createCheckBox $windowId -triState`;

// Set the value to "indeterminate"
setCheckBoxCheck $controlId 2;

// Get the value on the Check Box.
$checked = `getCheckBoxCheck $controlId`;
if( $checked == 1 )
{
    print "Checked";
}
else if( $checked == 0 )
{
    print "Unchecked";
}
else
{
    print "Indeterminate";
}
```

Additional information

Related commands

- [createCheckBox \(page 239\)](#)
- [getCheckBoxCheck \(page 465\)](#)
- [setCheckBoxHandler \(page 1035\)](#)

setCheckBoxHandler

Description

Summary

Sets the event handler for the user check box.

Details

Sets the event handler for the check box user control specified by `userControlId`. Event handlers are simply scripts to be called when certain events happen.

User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear a event handler, pass the empty string, "", to the flag.

If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setCheckBoxHandler userControlId [-click string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
click	1	string	—	Specifies the script to call when the user clicks on the check box.

Return value

void

Examples

```
// Set the event handler for the User Check Box.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createCheckBox $windowId -text "Check Me"`;
// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "CheckBoxId" $controlId;

// Set the -click event handler for the Check Box. When
// the user clicks on the Check Box, this script will be
// executed. CheckBoxClickHandler must reside in one of
// the scripts directories.
setCheckBoxHandler $controlId -click "CheckBoxClickHandler";
```

Additional information

Related commands

- [setColorPickerHandler \(page 1040\)](#)
- [setDropListHandler \(page 1058\)](#)
- [setListBoxHandler \(page 1088\)](#)
- [setNumBoxHandler \(page 1111\)](#)

-
- [setPushbuttonHandler \(page 1133\)](#)
 - [setRadioButtonHandler \(page 1138\)](#)
 - [setStaticBoxHandler \(page 1160\)](#)
 - [setTextBoxHandler \(page 1170\)](#)
 - [setTimeBoxHandler \(page 1175\)](#)
 - [setWindowHandler \(page 1188\)](#)

setColorPickerColor

Description

Summary

Set the user color picker color value.

Details

Sets the color value of the color picker user control specified by `userControlId`.

The second argument is a three element integer array, holding the Red, Green, and Blue components of the color to set on the control. Each component can range in value from 0 to 255.

Functional area

User Window

Command syntax

Syntax

```
setColorPickerColor userControlID colorIntArray
```

Arguments

Name	Type	Required	Comments
<code>intArray</code>	<code>int_array</code>	yes	Three element integer array holding the RGB color value to set on the control.
<code>userControlId</code>	<code>int</code>	yes	ID of user control to operate on.

Flags

None

Return value

void

Examples

```
// Set the color value of a Color Picker User Control
int $windowId;
int $controlId;
int $color[3];

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createColorPicker $windowId`;

// Set the default color to red
$color[ 0 ] = 255;
$color[ 1 ] = 0;
$color[ 2 ] = 0;
setColorPickerColor $controlId $color;
```

Additional information

Related commands

- [createColorPicker \(page 242\)](#)
- [getColorPickerColor \(page 475\)](#)
- [setColorPickerHandler \(page 1040\)](#)

setColorPickerHandler

Description

Summary

Sets the event handler for the user color picker.

Details

Sets the event handler for the color picker user control specified by `userControlId`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag.

If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setColorPickerHandler userControlId [-change string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for. Specifies the script to call when the user changes the color of the color picker.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
change	1	string	—	—

Return value

void

Examples

```
// Set the event handler for the User Color Picker.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createColorPicker $windowId`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "ColorPickerId" $controlId;

// Set the -change event handler for the Color Picker. When
// the user changes the color of the Color Picker, this script will be
// executed. ColorPickerChangeHandler must reside in one of
// the scripts directories.
setColorPickerHandler $controlId -change "ColorPickerChangeHandler";
```

Additional information

Related commands

- | [setCheckBoxHandler \(page 1035\)](#)
- | [setDropListHandler \(page 1058\)](#)
- | [setListBoxHandler \(page 1088\)](#)
- | [setNumBoxHandler \(page 1111\)](#)
- | [setPushButtonHandler \(page 1133\)](#)
- | [setRadioButtonHandler \(page 1138\)](#)
- | [setStaticBoxHandler \(page 1160\)](#)

-
- [setTextBoxHandler \(page 1170\)](#)
 - [setTimeBoxHandler \(page 1175\)](#)
 - [setWindowHandler \(page 1188\)](#)

setConstraint

Description

Summary

Sets the constraint offset for all selected nodes.

Details

The setConstraint command sets the constraint offset for all selected nodes based on the position and rotation of the nodes at the current frame. Selected nodes may be constraints or sources of constraints.

Used often during fine tuning the setup of a skeleton to adjust the relationship between markers and bones.

Functional area

Skeletal solving

Command syntax

Syntax

```
setConstraint [-setKey] [-solvingOnly] [-labelingOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
setKey	0	—	—	Sets the constraint offset as a key rather than a default value.
solvingOnly				
labelingOnly				

Return value

void

Examples

```
//With a marker selected, execute the setConstraint command to
// update the constraint offset
setConstraint;
```

Additional information

Related commands

- | [createSkelScript \(page 291\)](#)
- | [snapToConstraint \(page 1209\)](#)
- | [solve \(page 1226\)](#)

setControlAnchor

Description

Summary

Attaches the side of a user control to the side of another user control.

Details

Attaches the side of a user control to the side of another user control. When a user window's size changes, the user control's position and size (layout) on the window often needs to be recalculated. By attaching the sides of a user control to other user controls, the user control's layout will automatically be handled by the framework. This is easier than calling [setControlPos \(page 1048\)](#) on each user control every time the user window's size changes.

User controls can be anchored to the sides of forms and/or other user controls. For a description of forms, see [createForm \(page 250\)](#). To be anchored to the side of a form, the user control must have been created with the `-form` flag, and thus added to the form's list of user controls to manage. When anchoring to a form, `controlSide` and `targetSide` are typically the same (e.g. the left side of a control is anchored to the left side of a form. When the form's left side gets moved, the user control's left side is moved with it.).

Alternatively, a user control can be attached to another user control. This allows user controls to be laid out with respect to each other (e.g. the left side of a control can be anchored to the right side of another control. When the right side of the other control gets moved, the user control's left side will be moved as well.).

To dynamically control the size of a user control, anchor the control's opposing sides. If opposite sides of a user control are anchored, the control's size is also affected because it has anchors pulling its sides in opposite directions.

Note that `controlSide` and `targetSide` must specify either matching or opposing sides, but not adjacent sides. E.g. specifying "left" and "top" is illogical and will result in an error.

Functional area

User Window

Command syntax

Syntax

```
setControlAnchor userControlID "controlSide" "targetSide" offsetInt[-percent] [-target integer]
```

Arguments

Name	Type	Required	Comments
offset	int	yes	Offset in pixels or in percent if the percent flag is specified.
targetSide	string	yes	Side of the target user control which will be the anchor to the user control. Could be left,right, top, and bottom
controlSide	string	yes	Side of the user control to anchor to the target user control. Could be left, right, top, and bottom
controlId	int	yes	ID of user control whose side will be anchored. If none is specified,then the anchor will be to the side of the form

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
percent	0	—	—	—
target	1	integer	—	—

Return value

void

Examples

```
// Anchor two controls two each other, and to the
// form top-level
int $windowId;
int $staticId, $textId;
int $formId;
// First create a User Window to place the Controls
on$windowId = `createWindow "MyWindow"`;

// Get the top-level form
Id$formId = `getTopLevelForm $windowId`;

// Create a label User Control, and add it to the form
$staticId = `createStaticBox $windowId -text "Name" -form $formId`;

// Create a Text Box Control in the window. Also add it to the
// top-level form.
$textId = `createTextBox $windowId -form $formId`;

// Attach the Static User Control to the top and to the left
// of the top-level form
setControlAnchor $staticId "left" "left" 0;
setControlAnchor $staticId "top" "top" 0;

// Make the Text Box User Control stretch from the right side
// of the Static Box, to the right side of the User Window
setControlAnchor $textId "top" "top" 0 -target $staticId;
setControlAnchor $textId "left" "right" 5 -target $staticId;
setControlAnchor $textId "right" "right" 0;

// Issue the call to layout the top level form. We only
// need to do this once
layoutForm $formId;
```

Additional information

Related commands

- | [createForm \(page 250\)](#)
- | [getControlPos \(page 491\)](#)
- | [getTopLevelForm \(page 710\)](#)
- | [layoutForm \(page 770\)](#)
- | [setControlPos \(page 1048\)](#)

setControlPos

Description

Summary

Set the position of a user control.

Details

Sets the position of the user control specified by `userControlId`.

`intArray` holds the x,y coordinates of the 4 points of a rectangle, whose coordinates are relative to the top/left corner of the user window that the control is on.

You can explicitly lay out all of your user controls using this method, or by using the [setControlAnchor \(page 1045\)](#) command to have the user controls be dynamically sized /positioned for you. Using `setControlAnchor` is the preferred method, since the layout logic only needs to be specified once and the user controls will be sized/placed automatically any time the user window gets resized.

All user controls have a default size, and are created in the top left corner of the user window, so they must be placed by using either the `-pos` option when being created, using `setControlPos`, or `setControlAnchor`. Otherwise, all of the user controls are piled on top of one another in the top left corner of the user window.

Functional area

User Window

Command syntax

Syntax

```
setControlPos userControlID intArray[4]
```


Arguments

Name	Type	Required	Comments
intArray	int_array	yes	A four-element int array representing a rectangle, which specifies the initial size/position of the control relative to the top left corner of the user window. The element order is Left, Top, Right, and Bottom
userControlId	int	yes	ID of the user control to set position for.

Flags

None

Return value

void

Examples

```
// Position the control in the middle of the User Window
int $windowId;
int $controlId;
int $rect[4], $controlRect[4];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Text Box Control in the window.
$controlId = `createTextBox $windowId -text "I am centered"`;

// Get the rect of the User Window, so we can center the
// Text Box in it
$rect = `getWindowRect $windowId`;

// Now center it in the window
$controlRect[0] = $rect[0];
$controlRect[1] = ($rect[1] + $rect[3]) / 2;
$controlRect[2] = $rect[2];
$controlRect[3] = $controlRect[1] + 24;
setControlPos $controlId $controlRect;
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlText \(page 1051\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)
- | [showControl \(page 1195\)](#)

setControlText

Description

Summary

Set the text of a User Control.

Details

Sets the text on the user control specified by `userControlId`.

Each user control has its own text which is usually displayed with the control. This text is not to be confused with tooltip text, which only gets displayed when a user hovers the mouse over the user control. For text boxes, the control text is what the user edits. For other user control types, the text is for display only.

Some user controls, like color pickers, don't have any visible text, so this command doesn't have any effect on them. All user controls have no text by default, unless it is created using the `-text` option.

Functional area

User Window

Command syntax

Syntax

```
setControlText userControlID "controlText"
```

Arguments

Name	Type	Required	Comments
controlText	string	yes	Text to set for the user control
userControlId	int	yes	ID of user control to set text for.

Flags

None

Return value

void

Examples

```
// Set the text of a control.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Text Box Control in the window.
$controlId = `createTextBox $windowId`;

// Set the text
setControlText $controlId "Some Text";
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)
- | [showControl \(page 1195\)](#)

setControlTip

Description

Summary

Set the tooltip of a user control.

Details

Sets the tooltip text on the user control specified by `userControlID`.

Tooltips are text boxes that appear when a mouse is hovered over a user control. They are used to give the user feedback on the purpose and usage of a user control. They are not to be confused with the actual user control text.

All user controls have no tooltip text by default, unless created using the `-tip` option.

Functional area

User Window

Command syntax

Syntax

```
setControlTip userControlID "tipText"
```

Arguments

Name	Type	Required	Comments
tipText	string	yes	Tooltip text to set for the user control
userControlId	int	yes	ID of user control to set tooltip for.

Flags

None

Return value

void

Examples

```
// Set the tooltip text of a control.
int $windowId;
int $controlId;
int $rect[4];

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Give the control a decent size/position
$rect[0] = 20;
$rect[1] = 20;
$rect[2] = 100;
$rect[3] = 40;

// Create a Text Box Control in the window. We can either
// set the tip here, or set it using setControlTip.
$controlId = `createTextBox $windowId -pos $rect`;

// Set the tip
setControlTip $controlId "Enter your name";
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)

-
- [setControlText \(page 1051\)](#)
 - [setFocus \(page 1070\)](#)
 - [showControl \(page 1195\)](#)

setDir

Description

Summary

Specify the current working directory.

Details

Specify the current working directory. The current directory is used by some Windows system commands. By default the current directory is the Shogun Post applications root directory.

Functional area

System

Command syntax

Syntax

```
setDir "folderPath"
```

Arguments

Name	Type	Required	Comments
folderPath	string	yes	Path to serve as the current directory. This path must use forward slashes and may not be more than 260 characters long.

Flags

None

Return value

boolean

Returns true if the operation was successful.

Examples

```
// Set the new working directory to be our temp folder.  
boolean $wasSuccessful = `setDir "C:/Temp/"`;  
print $wasSuccessful;
```

setDropListHandler

Description

Summary

Sets the event handler for the user drop list.

Details

Sets the event handler for the drop list user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setDropListHandler userControlID [-selChange string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selChange	1	string	—	Specifies the script to call when the user changes the selection of the drop list.

Return value

void

Examples

```
// Set the event handler for the User Drop List.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createDropList $windowId "one" "two"`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "DropListId" $controlId;

// Set the -selChange event handler for the Drop List. When
// the user changes the selection of the Drop List, this script will be
// executed. DropListChangeHandler must reside in one of
// the scripts directories.
setDropListHandler $controlId -selChange "DropListChangeHandler";
```

Additional information

Related commands

- [setCheckBoxHandler \(page 1035\)](#)
- [setColorPickerHandler \(page 1040\)](#)
- [setListBoxHandler \(page 1088\)](#)

-
- [setNumBoxHandler \(page 1111\)](#)
 - [setPushButtonHandler \(page 1133\)](#)
 - [setRadioButtonHandler \(page 1138\)](#)
 - [setStaticBoxHandler \(page 1160\)](#)
 - [setTextBoxHandler \(page 1170\)](#)
 - [setTimeBoxHandler \(page 1175\)](#)
 - [setWindowHandler \(page 1188\)](#)

setEditTool

Description

Summary

Sets which editing tool is active.

Details

Sets the active editing tool type. Here is a list of editing tools and their type numbers:

Editing tool	Type number
New point	0
Linear	1
Bell	2
Flatten/Expand	3
Offset	4

Functional area

Interface

Command syntax

Syntax

```
setEditTool toolID
```

Arguments

Name	Type	Required	Comments
toolID	int	yes	ID of the Editing Tool to set active. See table above for valid values.

Flags

None

Return value

integer

Examples

```
// Get the current editing tool, and cycle to the next one.
int $tool, $next;

// Get the current one.
$tool = `getEditTool`;

// Figure out the next one.
$next = $tool + 1;
if( $next > 4 )
{
    $next = 0;
}
setEditTool $next;
```

Additional information

Related commands

- [getEditTool \(page 523\)](#)
- [graphView \(page 730\)](#)
- [manipulator \(page 789\)](#)

setErrorHandler

Description

Summary

Set a script to run when a script error has occurred.

Details

Shogun Post allows you to specify a script which gets automatically run if another script it is executing fails. This may be useful when setting up a farm of Shogun Post machines which automatically process files. The error handler script may, for instance, call the system command to run a Python script to email the interested parties.

Functional area

System

Command syntax

Syntax

```
setErrorHandler ["script"]
```

Arguments

Name	Type	Required	Comments
script	string	no	Name of script the to execute. Do not specify this argument, or pass an empty string, if you wish to clear the error handler.

Flags

None

Return value

void

Examples

```
// Set our custom error handler
setErrorHandler "MyErrorHandlerScript";

// Clear it
setErrorHandler "";
// Could also have called setErrorHandler
```


setExitCode

Description

Summary

Set the exit code passed back to the Windows system when Shogun Post shuts down.

Details

All processes in Windows return an exit code when the shut down. The exit code is often used to signify some sort of failure, but can signal anything.

By default Shogun Post returns 0, but you may wish it to return something else in certain situations.

Functional area

System

Command syntax

Syntax

```
setExitCode exitCode
```

Arguments

Name	Type	Required	Comments
exitCode	integer	yes	The exit code to be passed back to the Windows system when Shogun Post shuts down.

Flags

None

Return value

void

Examples

```
// Set the exit code to -5, which we agree to mean that this script
// that we rely on failed
if( `runScript "MyImportantScript; " -noFail` == false )
{
    setExitCode -5;
    exit;
}
```

Additional information

Related commands

- | [exit \(page 369\)](#)
- | [runScript \(page 941\)](#)

setExtrapolateFingersAfterSolve

Description

Set extrapolate fingers after solve.

Functional area

Skeletal solving

Command syntax

Syntax

```
setExtrapolateFingersAfterSolve boolean
```

Arguments

Name	Type	Required	Comments
boolVal	boolean		Boolean to turn on or off

Flags

None

Return value

void

Additional information

Related commands

- | [extrapolateFingers \(page 371\)](#)
- | [getExtrapolateFingersAfterSolve \(page 525\)](#)

setFilePos

Description

Summary

Set the current file position

Details

Use this command to set your position in a file. The value is in bytes from the origin, which you specify in the `origin` argument. The file position can be set past the end of the file in Write mode.

This command only makes sense in files opened in binary mode.

Functional area

Disk I/O

Command syntax

Syntax

```
setFilePos fileID offset [-cur] [-end]
```

Arguments

Name	Type	Required	Comments
<code>origin</code>	string		Origin that the offset is defined against, either beg for the beginning of the file, cur for the current file position, or end for the end of the file.
<code>offset</code>	int		Offset in bytes from the origin to set the file position to
<code>fileID</code>	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cur	0	—	end	—
end	0	—	cur	—

Return value

void

Examples

```
int $fileID;
int $filePos;
$fileID = `fileOpen "C:/testfile.log" "r"`;

// Read a value from the file, which "points" to another part of the
// file. This could be done in a file "header" section
$filePos = `readInt $fileID`;
print $filePos;

// Move to the file position "pointed" to earlier, so we can do some
// reading
setFilePos $fileID $filePos;
```

Additional information

Related commands

■ [getFilePos \(page 537\)](#)

setFocus

Description

Summary

Set the input focus to a user control.

Details

Sets the input focus on the user control specified by `userControlID`. Only one control (either user or native) can have the input focus at any one time.

The control with the input focus is the one that receives keyboard and mouse input. You can set the focus to a user control so that it will be the recipient of keyboard and mouse messages.

The input focus is lost when the user clicks outside of the control, or tabs outside of it.

Functional area

User Window

Command syntax

Syntax

```
setFocus userControlID
```

Arguments

Name	Type	Required	Comments
<code>userControlID</code>	int	yes	ID of user control to set focus on

Flags

None

Return value

void

Examples

```
// Set the focus to one of our controls.
int $windowId;
int $controlId;

// First create a User Window to place the control on
$windowId = `createWindow "MyWindow"`;

// Create a Text Box Control in the window.
$controlId = `createTextBox $windowId`;

// Set the focus to the control
setFocus $controlId;
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlAnchor \(page 1045\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlTip \(page 1053\)](#)
- | [showControl \(page 1195\)](#)

setFrameRate

Description

Summary

Set the Shogun Post system and scene rate.

Details

This command controls the hardware system and scene rates. It allows you to specify the rate standard to be one of the industry standard types (e.g. "ntsc" or "pal"), or to run at some arbitrary frame rate (using the "other" rate type). If you are connected to a Vicon system, calling this command will update the rate that the system runs at.

Functional area

System

Command syntax

Syntax

```
setFrameRate "rateType" fps
```

Arguments

Name	Type	Required	Comments
rateType	string	yes	Specify the frame rate type. Valid values are "ntsc", "ntscDrop", "pal", "film", "filmNtsc" and "other"
fps	float	yes	The frames per second of the scene rate. The rate will be rounded to the closest compatible rate for the given rate type (or standard). For instance, if rateType is "ntsc" and you specify "120" for the rate, the rate will get adjusted to 119.88. This rounding does not happen if the standard is "other".

Flags

None

Return value

void

Examples

```
// We want our base rate to be ntsc film (23.976)
// But the system rate to be a multiple of that.
setFrameRate "filmNtsc" 119.88;
```

Additional information

Related commands

■ [getTimecodeStandard \(page 708\)](#)

setGlobalVar

Description

Summary

Set global variable value

Details

Functional area

Data retrieval

Command syntax

Syntax

```
setGlobalVar name value
```

Arguments

Name	Type	Required	Comments
value	int	yes	Value to specify for variable
name	string	yes	Name of the variable to set

Flags

None

Return value

string

setHotKey

Description

Summary

Assigns a hot key to a script or command.

Details

Sets a hot key for a command or script. If the hot key combination is already assigned to another command, it will be reassigned to this new one.

Windows does not provide a way to individually remove hot keys for an application (it only provides a way to replace the list of hot keys). Therefore, Shogun Post keeps an internal list of all hot keys, until the `-final` flag is specified, when it copies the internal list to the application's list. This is done to speed up execution of the command.

The second argument (key) requires some special handling:

- For letters, use the letter itself (e.g. "s" for the S key).
- For numbers and other single character keys (e.g. "="), use the character itself, but be sure to enclose it in quotes (e.g. "=").

Here is a table of all other special keys, and the string that is expected for the second argument:

Key	String
Backspace	"BACK"
Tab	"TAB"
Enter	"ENTER" or "RETURN"
Pause/Break	"PAUSE"
Caps Lock	"CAPS"
Escape	"ESC"
Page Up	"PAGEUP"
Page Down	"PAGEDOWN"

Key	String
Home	"HOME "
End	"END "
Left Arrow	"LEFT "
Right Arrow	"RIGHT "
Up Arrow	"UP "
Down Arrow	"DOWN "
Select	"SEL "
Execute	"EXECUTE "
Print Screen	"PRINTSCREEN "
Insert	"INSERT "
Delete	"DELETE "
Help	"HELP "
Num Lock	"NUMLOCK "
Scroll Lock	"SCROLLLOCK "
F1-F12	"F1 " - "F12 "

Functional area

System

Command syntax

Syntax

```
setHotKey ["commandName" "key"] [-ctrl] [-shift] [-numPad] [-final]
```

Arguments

Name	Type	Required	Comments
commandName	string	yes	Name of a native command or script which will execute when the Hot Key is pressed. If a script, the script must exist in the scripts directories specified in the Preferences.
key	string	yes	Key on the keyboard specifying the hot key. See <i>Description</i> above for valid values.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ctrl	0	—	—	Specifies that the CTRL key must also be depressed for the command to execute. The default is no modifier.
shift	0	—	—	Specifies that the Shift key must also be pressed for the command to execute. The default is no modifier.
numPad	0	—	—	Certain keys are repeated on the number pad portion of keyboards (e.g. 19, +, , etc). Specifies that the key is the one on the number pad.
final	0	—	—	Specifies that this call to setHotKey is the last in a series. This flag should always be provided if only one call to setHotKey is being made. This is to reduce allocation and reallocation of hot keys in the system.

Return value

void

Examples

```
// Set some hot keys. Be sure to specify the -final
// flag on the last one.
setHotKey "cutKeys" "x" -ctrl;
setHotKey "fillGaps" "f" -shift;
setHotKey "step" "RIGHT";
setHotKey "play" "TAB";
setHotKey "stop" "TAB" -ctrl;
setHotKey -final;
```

Additional information

Related commands

- | [createShelfButton \(page 282\)](#)
- | [createShelfSeparator \(page 287\)](#)
- | [createShelfTab \(page 289\)](#)
- | [deleteShelfTab \(page 350\)](#)
- | [getNumShelfTabs \(page 625\)](#)
- | [renameShelfTab \(page 925\)](#)

setHotKeyFile

Description

Summary

Set the file where Shogun Post loads and saves your hot keys.

Details

Set the file where Shogun Post loads and saves your hot keys.

Functional area

System

Command syntax

Syntax

```
setHotKeyFile HotKeyFile [-save]
```

Arguments

Name	Type	Required	Comments
HotKeyFile	string	yes	Full path of the file where Shogun Post loads and saves your hot keys. Use forward slashes instead of backslashes for all file paths.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
save	0	—	—	Causes the current hot keys to be saved to the given file. If not specified, then the hot keys in the file will be loaded.

Return value

void

Examples

```
// Save your existing hot keys to the root of your C: drive
setHotKeyFile "C:/MyHotKeys.hsl" -save;
```

Additional information

Related commands

- [setAutoSaveFile \(page 1027\)](#)
- [setLogFile \(page 1104\)](#)
- [setMarkingMenuFile \(page 1109\)](#)

setInterpType

Description

Summary

Set the interpolation type being used between keys.

Details

By default, Shogun Post uses a Bezier curve to determine values between keys.

Functional area

Data manipulators

Command syntax

Syntax

```
setInterpType propertyName interpType [-onMod string]
```

Arguments

Name	Type	Required	Comments
interpType	string	yes	Type of interpolation to be used. This applies to properties of type Float (Step, Linear, Bezier), Rotation (Step, Linear, Cubic, Bezier) and Vec3 (Step, Linear, Bezier).
propertyName	string	yes	Channel property on which the interpolation type is being set.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	This will apply to a single module at the path string location. Otherwise the change applies to all selected modules.

Return value

void

Examples

```
select LFWT;
setInterpType Translation Step;
```

setKey

Description

Summary

Set a key on the named property of the selected modules.

Details

Use setKey to set explicit key frames on the named property. These values can either be absolute or relative to existing values.

Functional area

Data manipulators

Command syntax

Syntax

```
setKey "propertyType" value1 ...[-t integer] [-r] [-all] [-onMod string]
[-type string]
```

Arguments

One string argument indicating the property to be keyed, one or three float arguments, dependent upon the property name called. For example, one value for TranslationX, and three values if Translation is indicated. Set value relative to current value instead of absolute value

Name	Type	Required	Comments
floatValue3	float		Property value3 if applicable, depends on the property
floatValue2	float		Property value2 if applicable, depends on the property
floatValue1	float		Property value1

Name	Type	Required	Comments
propertyName	string		String name of the property for the key

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
t	1	integer	—	—
r	0	—	—	—
onMod	1	string	—	—
-all				
type		string		

Return value

void

Examples

```
select LFWT;
  setKey -t 250 "Translation" 35 15 150;
// Set a key for the Translation XYZ property of the Marker "LFWT"
// at frame 250.
```

Additional information

Related commands

■ [setProperty \(page 1130\)](#)

setLanguage

Description

Details

Sets the language used from the Shogun Post command line. You can set the language to Python or HSL.

Functional area

System

Command syntax

Syntax

```
setLanguage language
```

Arguments

Name	Type	Required	Comments
language		yes	Can be Python or HSL.

Flags

None

Return value

boolean

Examples

```
setLanguage "python";  
// Sets Shogun Post command line language to Python.
```

setLength

Description

Summary

Returns a scaled vector of length len, collinear with vec. Does not modify vec.

Details

Use setLength when you need to set the length of a given vector. For instance, you might want to change the length of a given bone, which can be thought of as vector from one bone joint to another.

The length of the vector is the square root of the sum of the squares of each vector component.

Useful for setting the lengths of the bones in a skeleton based on performer dimensions.

Note that the input vector vector is not modified.

Functional area

Math

Command syntax

Syntax

```
setLength vector length
```

Arguments

Name	Type	Required	Comments
vector	vector	yes	Vector that points in the desired direction
length	float	yes	Float value of vector length or magnitude

Flags

None

Return value

vector

Returns a variable of type vector which points in the same direction as vec, but is of length len.

Examples

```
// Declare a vector, then changes its length
// The command will return a vector value of: <>
vector $v1 = <<1, 1, 0>>, $v1 = `setLength $v1 1.0`;
print( $v1 );
```

Additional information

Related commands

- [dot \(page 363\)](#)
- [getLength \(page 576\)](#)

setListBoxHandler

Description

Summary

Sets the event handler for the user list box.

Details

Sets the event handler for the list box user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them.

Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear a event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setListBoxHandler userControlID [-selChange string] [-dblClick string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selChange	1	string	—	Specifies the script to call when the user double clicks on a list box Item.
dblClick	1	string	—	—

Return value

void

Examples

```
// Set the event handler for the User List Box.int
$windowId;int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createListBox $windowId "1" "2"`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "ListBoxId" $controlId;

// Set the -dblClick event handler for the list box. When
// the user double-clicks on a list box item, this script will be
// executed. ListBoxDbClickHandler must reside in one of
// the script's directories.
setListBoxHandler $controlId -dblClick "ListBoxDbClickHandler";
```

Additional information

Related commands

- [setCheckBoxHandler \(page 1035\)](#)
- [setColorPickerHandler \(page 1040\)](#)
- [setDropListHandler \(page 1058\)](#)

-
- | [setNumBoxHandler \(page 1111\)](#)
 - | [setPushButtonHandler \(page 1133\)](#)
 - | [setRadioButtonHandler \(page 1138\)](#)
 - | [setStaticBoxHandler \(page 1160\)](#)
 - | [setTextBoxHandler \(page 1170\)](#)
 - | [setTimeBoxHandler \(page 1175\)](#)
 - | [setWindowHandler \(page 1188\)](#)

setListViewColumns

Description

Summary

Set the columns that a list view has

Details

This command allows you to specify the columns in a list view user control. You must call this before adding any items to the list view. You may only call this once. Subsequent calls will fail.

Functional area

User Window

Command syntax

Syntax

```
setListViewColumns userControlID "columnNameArray" [columnWidthArray]
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the list view to operate on
columnNameArray	string array	yes	A string array containing the names of the columns in the list view.
columnWidthArray	integer array	no	An integer array containing the width of each column. This argument is optional. However, if provided, the size of the array must match the size of columnNameArray.

Flags

None

Return value

void

Examples

```
// Demonstrate usage of a list view user control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columnsstring
$columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the first
// column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
```

```
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [deleteListViewItem \(page 342\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

setListViewHandler

Description

Summary

Set script handlers for various list view events.

Details

This command allows you to specify script handlers for list view events. Handlers are scripts of your choosing that get called when an event happens, such as the checked state of an item changing.

Calling this command without specifying any of the flags has the effect of clearing all of the handlers for the list view.

Functional area

User Window

Command syntax

Syntax

```
setListViewHandler userControlID[-selChange string] [-checkChange string]
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the user control to operate on

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selChange	1	string	—	Specify the script to be called when the selection state of items in the list view changes.
checkChange	1	string	—	Specify the script to be called when the checked state of items in the list view changes.

Return value

void

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}
// Create window and list view
$windowID = `createWindow "ListViewTesting"`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;

// Create the columnsstring $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Set the script we want to get called when the selection state of
// one of the items changes
setListViewHandler -selChange "MySelChangeHandler";
```

```
// Save the value of the list control so we can get it our handler script
setGlobalVar "MyListViewID" $listViewID;
//
////////////////////////////////////
// In "MySelChangeHandler. This script must exist under one
// of your script directories.
int $listViewID = `getGlobalIntVar "MyListViewID"`;
print( `getListViewSelItems $listViewID` );
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [deleteListViewItem \(page 342\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewItemCheck \(page 1097\)](#)
- | [setListViewItemText \(page 1100\)](#)

setListViewItemCheck

Description

Summary

Set the checked state of a list view item.

Details

This command allows you to set the checked state of a list view item. Only list views created with the `-checkBoxes` flag should call this.

Functional area

User Window

Command syntax

Syntax

```
setListViewItemCheck userControlID row checked
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the list view item to operate on
row	integer	yes	Zero-based index of the row whose checked state will be set
checked	boolean	yes	The new checked state.

Flags

None

Return value

void

Examples

```
// Demonstrate usage of a List View User Control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}
// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkBoxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;
// Create the columnsstring $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the first
// column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
```

```
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob`s name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [deleteListViewItem \(page 342\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewItemText \(page 1100\)](#)

setListViewItemText

Description

Summary

Set the text of a list view item

Details

This command allows you to set the text of list view items. If the list view has multi columns, you can use this command to set the values of individual "cells".

Functional area

User Window

Command syntax

Syntax

```
setListViewItemText userControlID row column "itemStr"
```

Arguments

Name	Type	Required	Comments
userControlID	integer	yes	ID of the list view to operate on
row	integer	yes	Zero based index of the item's row whose text is to be set
column	integer	yes	Zero based index of the item's column whose text is to be set
itemStr	string	yes	The value of the list view item

Flags

None

Return value

void

Examples

```
// Demonstrate usage of a list view user control
int $windowID, $listViewID, $formID;

// Destroy window if it already exists
if( `windowExists "ListViewTesting"` == true )
{
    destroyWindow "ListViewTesting";
}

// Create window and list view and position them
$windowID = `createWindow "ListViewTesting"`;
$formID = `getTopLevelForm $windowID`;
$listViewID = `createListView $windowID -form $formID -checkboxes`;
setControlAnchor $listViewID "left" "left" 3;
setControlAnchor $listViewID "top" "top" 3;
setControlAnchor $listViewID "right" "right" 3;
setControlAnchor $listViewID "bottom" "bottom" 3;

// Create the columns
string $columns[3];
int $widths[3];
$columns[0] = "Name";
$columns[1] = "Age";
$columns[2] = "Gender";
$widths[0] = 150;
$widths[1] = 50;
$widths[2] = 100;
setListViewColumns $listViewID $columns $widths;

// Add some items to the list view. The text we supply is for the first
// column.
// We supply text for subsequent columns using setListViewItemText
addListViewItem $listViewID "Bob";
addListViewItem $listViewID "Mary";
addListViewItem $listViewID "Jim";
addListViewItem $listViewID "Ann";

// Set additional details
setListViewItemText $listViewID 0 1 "50";
```

```
setListViewItemText $listViewID 0 2 "Male";
setListViewItemText $listViewID 1 1 "32";
setListViewItemText $listViewID 1 2 "Female";
setListViewItemText $listViewID 2 1 "21";
setListViewItemText $listViewID 2 2 "Male";
setListViewItemText $listViewID 3 1 "44";
setListViewItemText $listViewID 3 2 "Female";

// Set the check box to true for the males.
setListViewItemCheck $listViewID 0 true;
setListViewItemCheck $listViewID 2 true;

// Change Bob's name
setListViewItemText $listViewID 0 0 "Ken";

// Select Mary and Jim
selectListViewItem $listViewID 2 true;
selectListViewItem $listViewID 1 true;

// Print out some of what we just did
print( `getListViewSelItems $listViewID` );
print( `getListViewItemText $listViewID 1 2` );
print( `getListViewItemCheck $listViewID 1` );
print( `getListViewItemCheck $listViewID 0` );
layoutForm $formID;
```

Additional information

Related commands

- | [addListViewItem \(page 113\)](#)
- | [createListView \(page 262\)](#)
- | [deleteAllListViewItems \(page 328\)](#)
- | [deleteListViewItem \(page 342\)](#)
- | [getListViewItemCheck \(page 583\)](#)
- | [getListViewSelItems \(page 589\)](#)
- | [getNumListViewItems \(page 620\)](#)
- | [selectListViewItem \(page 991\)](#)
- | [setListViewHandler \(page 1094\)](#)
- | [setListViewColumns \(page 1091\)](#)
- | [setListViewItemCheck \(page 1097\)](#)

setLogEnabled

Description

Summary

Enables or disables the log.

Functional area

System

Command syntax

Syntax

```
setLogEnabled enabled
```

Arguments

See above syntax.

Flags

None

Return value

void

Additional information

Related commands

■ [setLogFile \(page 1104\)](#)

setLogFile

Description

Summary

Specify the command logging file

Details

Specify the command logging file. The command log file is where all Shogun Post command feedback gets saved.

Functional area

System

Command syntax

Syntax

```
setLogFile path [-noFeedback] [-autoName]
```

Arguments

Name	Type	Required	Comments
path	string	no	The full path to the log file. If you do not specify this, the command log will default to being <i>CommandLog.log</i> in the <i>Logs</i> folder in your Shogun Post directory.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
noFeedback	0	—	—	Prevents the new log path from being echoed to the command log.
autoName	0	—	—	Automatically names the log based on the current time and the computer name that Shogun Post is running on. Useful when setting up your own farming system and want to identify a command log with a machine and a script execution.

Return value

void

Examples

```
// Automatically set the log file, which will be put
// in the Shogun Post app folder, under the Logs folder.
setLogFile -autoName;

// Set it to something very specific
setLogFile "c:/mylogfile.log";
```

Additional information

Related commands

- [setAutoSaveFile \(page 1027\)](#)
- [setHotKeyFile \(page 1079\)](#)
- [setMarkingMenuFile \(page 1109\)](#)

setMarkerConnection

Description

Summary

Sets the connecting line between two markers in the scene.

Details

Establishes a connecting line between two markers in the scene.

If the two markers already have a connecting line, the color of the line is updated (if different).

If multiple markers by the given names exist (e.g. when two characters are in the scene), the markers parented under the Priority Search module are connected.

To set connecting lines for multiple characters in the scene, run your connecting script after setting the Priority Search module to each character.

Connecting lines are used for visual grouping of markers and have no intrinsic affect on the marker data.

Functional area

Data editing

Command syntax

Syntax

```
setMarkerConnection "sourceNode" "targetNode" [-color integer integer  
integer] [-priorityOnly]
```

Arguments

Name	Type	Required	Comments
targetMarker	string	yes	Name of the marker to draw the line to.
sourceMarker	string	yes	Name of the marker to draw the line from.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
color	3	integer	—	Set the RGB color of the marker connection
priorityOnly	0	—	—	Set the connection between markers parented to the priority search module only.

Return value

void

Examples

```
// Draw connecting lines around the four head markers of each
// character in the scene.
int $i, $count;
string $chars[] = `getModules -type Character`;

// Get the number of characters in the scene
$count = `getCount $chars`;

// Iterate through and set the connections for each.
for( $i = 0;
    $i < $count; $i += 1 )
{
    // Set the Priority Search Module. Do this by first
    // setting it as the primary selection, then calling
    // setPriority
    setPrimary $chars[ $i ];
    setPriority;
```

```
// Now set the connections
setMarkerConnection "LFHD" "LBHD" -color 0 255 0;
setMarkerConnection "LBHD" "RBHD" -color 0 255 0;
setMarkerConnection "RBHD" "RFHD" -color 0 255 0;
setMarkerConnection "RFHD" "LFHD" -color 0 255 0;
}
```

Additional information

Related commands

- [getMarkerConnection \(page 594\)](#)
- [removeMarkerConnection \(page 909\)](#)

setMarkingMenuFile

Description

Summary

Set the file where Shogun Post loads and saves your marking menus.

Functional area

System

Command syntax

Syntax

```
setMarkingMenuFile MarkingMenuFile[-save]
```

Arguments

Name	Type	Required	Comments
MarkingMenuFile	string	yes	Name of the file where Shogun Post loads and saves your marking menus. Use forward-slashes instead of back-slashes for all file paths.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
save	0	—	—	Causes the current marking menus to be saved to the given file. If not specified, then the marking menus in the file will be loaded.

Return value

void

Examples

```
// Save your marking menus to the root of your C: drive  
setMarkingMenuFile "C:/MyMarkingMenus.hsl" -save;
```

Additional information

Related commands

- | [setAutoSaveFile \(page 1027\)](#)
- | [setHotKeyFile \(page 1079\)](#)
- | [setLogFile \(page 1104\)](#)

setNumBoxHandler

Description

Summary

Sets the event handler for the user num box.

Details

Sets the event handler for the num box user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument: a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setNumBoxHandler userControlID [-change string] [-loseFocus string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
change	1	string	—	Specifies the script to call when the user changes the value of the num box.
loseFocus	1	string	—	Specifies the script to call when the num box loses the input focus.

Return value

void

Examples

```
// Set the event handler for the User Num Box.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createNumBox $windowId`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "NumBoxId" $controlId;

// Set the -loseFocus event handler for the Num Box. When
// the user clicks (or Tabs) out of Num Box item, this script will be
// executed. NumBoxLoseFocusHandler must reside in one of
// the scripts directories.
setNumBoxHandler $controlId -loseFocus "NumBoxLoseFocusHandler";
```

Additional information

Related commands

- [setCheckBoxHandler \(page 1035\)](#)
- [setColorPickerHandler \(page 1040\)](#)
- [setDropListHandler \(page 1058\)](#)
- [setListBoxHandler \(page 1088\)](#)
- [setPushButtonHandler \(page 1133\)](#)
- [setRadioButtonHandler \(page 1138\)](#)
- [setStaticBoxHandler \(page 1160\)](#)
- [setTextBoxHandler \(page 1170\)](#)
- [setTimeBoxHandler \(page 1175\)](#)
- [setWindowHandler \(page 1188\)](#)

setNumBoxNum

Description

Summary

Set the user num box number value.

Details

Sets the number that appears in a num box user control specified by `userControlID`.

Num boxes are text boxes that restrict their input and output to numerical values. The number expected is either a float or an integer. If [createNumBox \(page 266\)](#) was used with the `-flt` flag, then all values are interpreted as floats. Otherwise, values are rounded down to the nearest integer (i.e. any decimal value will be truncated).

Functional area

User Window

Command syntax

Syntax

```
setNumBoxNum userControlID numVal
```

Arguments

Name	Type	Required	Comments
numVal	float	yes	Number value to set for the num box. An int can be used as well, and any cast will automatically be made by the num box
userControlId	int	yes	ID of user control to operate on.

Flags

None

Return value

void

Examples

```
// Set the value on a Num Box User Control
int $windowId;
int $controlId;
float $val;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createNumBox $windowId`;

// Set the value on the Num Box. Because the Num Box
// was not created with the -flt flag, we are really
// setting the value to be -10.
setNumBoxNum $controlId -10.75;
// Should be -10
$val = `getNumBoxNum $controlId`;
print $val;
```

Additional information

Related commands

- | [createNumBox \(page 266\)](#)
- | [getNumBoxNum \(page 610\)](#)
- | [setNumBoxHandler \(page 1111\)](#)

setNumBoxRange

Description

Summary

Set the range of a number box user control on the given user window.

Details

Set the range of a number box user control on the given user window so that there is a set minimum and maximum value that can not exceeded.

Functional area

User Window

Command syntax

Syntax

```
setNumBoxRange userControlID minVal maxVal
```

Arguments

Name	Type	Required	Comments
maxVal	float/int	yes	Maximum number that the num box will display
minVal	float/int	yes	Minimum number that the num box will display
userControlId	int	yes	ID of user control to apply command to

Flags

None

Return value

void

Examples

```
// Create a Check Box User Control.
int $windowId;
int $controlId;
float $value = 10.0;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the Control on the Window, passing
// in the Window Id of the User Window we
// just created. Make floating point, specifying the initial
// value and a spinner
$controlId = `createNumBox $windowId -num $value -spin -flt`;
setNumBoxRange $controlId 10.0 20.0;
```

Additional information

Related commands

- [createNumBox \(page 266\)](#)
- [getNumBoxNum \(page 610\)](#)

setParameter

Description

Summary

Sets a new value for a dynamic parameter.

Details

Given a parameter name, if the parameter exists on the selected character or a character is specified with the `-onMod` flag, the parameter value is set to the new value specified in the second argument. For dynamic parameters only.

Functional area

Parameters

Command syntax

Syntax

```
setParameter parameterName parameterValue[-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	String	Yes	Defines the name of the parameter.
parameterValue	Float	Yes	Defines the new value for the parameter.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	Defines the character that the parameter should be found on.
solving				

Return value

void

Examples

```
// Set the parameter named UpperArmLength to 200.
setParameter "UpperArmLength" 200.0;
```

Additional information

Related commands

- | [addParameter \(page 119\)](#)
- | [getParameter \(page 627\)](#)
- | [getParameters \(page 633\)](#)
- | [hasParameter \(page 737\)](#)
- | [listParameters \(page 777\)](#)
- | [removeAllParameters \(page 899\)](#)
- | [removeParameter \(page 913\)](#)
- | [removeUnusedParameters \(page 919\)](#)
- | [renameParameter \(page 921\)](#)
- | [selectByParameter \(page 968\)](#)

setParameterPrior

Description

Summary

Sets the prior value of a parameter of the given name.

Details

Functional area

Parameters

Command syntax

Syntax

```
setParameterPrior parameterName prior [-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	String	Yes	Defines the name of the parameter.
prior			

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod	1	string	—	May be used to specify the character on which the parameter exists. Without this option, the selected character is used.
solving				Specifies if the parameter exists on the solving setup instead of the labeling setup.

Return value

void

setPinned

Description

Summary

Pins the specified window (panel) as a tab along an edge of the main Shogun Post window.

Details

A pinned panel is displayed as a tab along an edge of the main window. The edge that the panel is pinned to depends upon the docked location of the specified panel, and can be top, bottom, left or right.

Functional area

Interface

Command syntax

Syntax

```
setPinned "windowName" boolean
```

Arguments

Name	Type	Required	Comments
windowName		yes	Identifier of panel to pin.

Flags

None

Return value

void

setPosition

Description

Summary

Details

Use setPosition to set the translation of a module in relative or world space coordinates.

setPosition modifies the keys of the current frame. If no animation data exists when setPosition is run, it creates a key.

Functional area

Data manipulators

Command syntax

Syntax

```
setPosition "moduleName" posVec [-ws] [-preTrans] [-default]
```

Arguments

Name	Type	Required	Comments
posVec			The vector data to apply
moduleName			Name of the module you want to modify

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	0	—	—	PosVector data is applied in world space coordinates
preTrans	0	—	default	—
default	0	—	preTrans	—

Return value

void

Examples

```
//Create a marker and set its position
create Marker "marker1";
setPosition marker1 <<100, 0, 0>>;
```

Additional information

Related commands

- | [setKey \(page 1083\)](#)
- | [setRotation \(page 1143\)](#)
- | [setScale \(page 1147\)](#)

setPreferredPose

Description

Summary

Sets the selected labeling or solving bones to their preferred pose.

Details

Functional area

Skeletal solving

Command syntax

Syntax

```
setPreferredPose
```

Arguments

None

Flags

None

Return value

void

setPrimary

Description

Summary

Set named module as primary selection.

Details

Use setPrimary to make any scene module the primary selection, whether or not that module is currently selected. If there is a primary selection at the time at which the command is executed, that module will remain selected, but will no longer be primary.

Functional area

Selection

Command syntax

Syntax

```
setPrimary "name" [-prev] [-next]
```

Arguments

Name	Type	Required	Comments
name	string	no	String name of the module to be declared as primary

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
prev	0	—	next	If multiple modules are selected, sets the primary module to the next module of those selected.
next	0	—	prev	If multiple modules are selected, sets the primary module to the previous module of those selected.

Return value

void

Examples

```
// Set "LFIN" to the primary selection.
// Currently selected modules remain selected.
setPrimary "LFIN";
```

Additional information

Related commands

■ [select \(page 956\)](#)

setPriority

Description

Summary

This command is used to change the current priority node.

Details

Select an object in the scene and execute setPriority. The selected object is now the priority module. If several objects are selected prior to issuing the command, the primary selection among them will be the new priority node.

Usually, the priority object is a CharacterNode so that you do not have to declare explicit paths for objects that are part of the CharacterNode hierarchical reading motion data into Shogun Post, the first CharacterNode is set by default as the priority node.

Functional area

Data manipulators

Command syntax

Syntax

```
setPriority
```

Arguments

None

Flags

None

Return value

void

Examples

```
select Actor_1;
setPriority;
select lfoot;
// Selects the left foot bone on Actor_1
select -priority;
// Selects Actor_1
select Actor_2;
setPriority;
select lfoot;
// Selects the left foot bone on Actor_2
```

Additional information

Related commands

- [select \(page 956\)](#) -priority
- [setPrimary \(page 1126\)](#)

setProperty

Description

Summary

Set the value of the selected keys on the named property on the selected modules.

Details

Use `setProperty` when you need to set the same value (in either absolute terms, or in terms relative to the existing keyframes) on a named property for the selected group of keys, whether the selection contains one object or several objects.

This command functions on attributes as well as channels. In the case of attributes, the current key selection is ignored since in Shogun Post, attributes are fixed settings and cannot be animated.

Functional area

Data manipulators

Command syntax

Syntax

```
setProperty propertyName propertyVal1 propertyVal2 ... [-r] [-all] [-onMod string] [-type string] [-d]
```

Arguments

Name	Type	Required	Comments
propertyName			<p>An argument indicating the property to be changed, usually followed by one to three arguments values.</p> <p>The number and type of argument values depends on the property called, eg, one numeric value float will be required if Translation.X is indicated, and three values if Translation is indicated. On the other hand, one string value is required for a property name like Name.</p> <p>The arguments can be taken directly from the names of the fields in the Attributes and Channels dialog boxes. For instance, Name and Rotation_Order are legitimate string arguments for Character Node attributes while Weight and Translation.X are legitimate float channel arguments for Direction Constraints and Markers respectively.</p>

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
c	0	—	—	Alter only the constant value for the property. Will not be visible to the user if keys already exist.
r	0	—	—	Set value relative to current value instead of absolute value. Requires keys to be selected.
all	0	—	—	Applies property change to all modules
onMod	1	string	type	Applies property change only to specified Module
type	1	string	onMod	Applies property change only to objects of a certain type
d	0	—	—	—

Return value

void

Examples

```
select LFWT LMWT LBWT RFWT RMWT RBWT;
selectRange 0 1000;
selectKeys -ranges;
setProperty -r "Translation" 0.0 0.0 400.0;
// For the range of selected keys, move the named nodes 40 millimeters
// in the Z direction relative to the selected keyframe values.

setProperty "Rotation_Order" YZX;
// Changes the rotation order to YZX for the selected object(s)
select LFHD;
setProperty "Name" taco;
// changes the name of the object called LFHD to taco;
```

Additional information

Related commands

■ [setKey \(page 1083\)](#)

setPushButtonHandler

Description

Summary

Sets the event handler for the User Push Button.

Details

Sets the event handler for the push button user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear a event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setPushButtonHandler userControlID [-click string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
click	1	string	—	Specifies the script to call when the user clicks on the push button.

Return value

void

Examples

```
// Set the event handler for the User Push Button.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createPushButton $windowId -text "Push"`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "PushButtonId" $controlId;

// Set the -click event handler for the Push Button. When
// the user mouse clicks on the Push Button, this script will be
// executed. PushButtonClickHandler must reside in one of
// the scripts directories.
setPushButtonHandler $controlId -click "PushButtonClickHandler";
```

Additional information

Related commands

- [setCheckBoxHandler \(page 1035\)](#)
- [setColorPickerHandler \(page 1040\)](#)
- [setDropListHandler \(page 1058\)](#)
- [setListBoxHandler \(page 1088\)](#)

-
- | [setNumBoxHandler \(page 1111\)](#)
 - | [setRadioButtonHandler \(page 1138\)](#)
 - | [setStaticBoxHandler \(page 1160\)](#)
 - | [setTextBoxHandler \(page 1170\)](#)
 - | [setTimeBoxHandler \(page 1175\)](#)
 - | [setWindowHandler \(page 1188\)](#)

setRadioButtonCheck

Description

Summary

Set the user radio button check value.

Details

Sets the check value of the radio button user control specified by `userControlID`.

Radio buttons are essentially check boxes, but they operate in a group with other radio buttons, allowing for mutually exclusive choices. When a user interacts with radio buttons in a group, the radio buttons automatically uncheck themselves when another button in the group is checked. However, when setting the check with this command, you must explicitly uncheck any other radio buttons in the group, otherwise, multiple buttons may be checked.

Functional area

User Window

Command syntax

Syntax

```
setRadioButtonCheck userControlID checkBoolean
```

Arguments

Name	Type	Required	Comments
checkVal	boolean	yes	The check value of this radio button.
userControlId	int	yes	ID of user control to operate on.

Flags

None

Return value

boolean

Examples

```
// Set the checked value of a Radio Button User Control
int $windowId;
int $controlId1, $controlId2, $controlId3;
int $rect[ 4 ];

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Create the User Controls on the Window. Check the middle one.
$rect[ 0 ] = 20;
$rect[ 1 ] = 20;
$rect[ 2 ] = 100;
$rect[ 3 ] = 40;
$controlId1 = `createRadioButton $windowId -pos $rect -text "Radio 1"`;
$rect[ 1 ] = 50;
$rect[ 3 ] = 70;
$controlId2 = `createRadioButton $windowId -pos $rect -text "Radio 2" -check`;
$rect[ 1 ] = 80;
$rect[ 3 ] = 100;
$controlId3 = `createRadioButton $windowId -pos $rect -text "Radio 3"`;

// Now check the last one, making sure to uncheck the middle one we
// checked before
setRadioButtonCheck $controlId1 false;
setRadioButtonCheck $controlId2 false;
setRadioButtonCheck $controlId3 true;
```

Additional information

Related commands

- [createRadioButton \(page 274\)](#)
- [getRadioButtonCheck \(page 657\)](#)
- [setRadioButtonHandler \(page 1138\)](#)

setRadioButtonHandler

Description

Summary

Sets the event handler for the user radio button.

Details

Sets the event handler for the radio button user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear a event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setRadioButtonHandler userControlID[-click string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
click	1	string	—	Specifies the script to call when the user clicks on the radio button.

Return value

void

Examples

```
// Set the event handler for the User Radio Buttons.
int $windowId;
int $radio1, $radio2, $radio3;
int $rect[4];

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Create the User Controls on the Window.
$rect[0] = 20;
$rect[1] = 20;
$rect[2] = 80;
$rect[3] = 40;
$radio1 = `createRadioButton $windowId -pos $rect -text "Option1"`;
$rect[0] = 20;
$rect[1] = 50;
$rect[2] = 80;
$rect[3] = 70;
$radio2 = `createRadioButton $windowId -pos $rect -text "Option2"`;
$rect[0] = 20;
$rect[1] = 80;
$rect[2] = 80;
$rect[3] = 100;
$radio3 = `createRadioButton $windowId -pos $rect -text "Option3"`;

// Save the control IDs to our profile, so we can retrieve them
// from the event handler
writeProfileInt "MyWindowSection" "RadopButton1Id" $radio1;
writeProfileInt "MyWindowSection" "RadopButton2Id" $radio2;
writeProfileInt "MyWindowSection" "RadopButton3Id" $radio3;
```

```
// Set the -click event handler for the Radio Buttons. When
// the user mouse clicks on any of the Radio Buttons,
// this script will be executed. RadioButtonClickHandler
// must reside in one of the scripts directories.
setRadioButtonHandler $radio1 -click "RadioButtonClickHandler";
setRadioButtonHandler $radio2 -click "RadioButtonClickHandler";
setRadioButtonHandler $radio3 -click "RadioButtonClickHandler";
```

Additional information

Related commands

- | [setCheckedHandler \(page 1035\)](#)
- | [setColorPickerHandler \(page 1040\)](#)
- | [setDropListHandler \(page 1058\)](#)
- | [setListBoxHandler \(page 1088\)](#)
- | [setNumBoxHandler \(page 1111\)](#)
- | [setPushButtonHandler \(page 1133\)](#)
- | [setStaticBoxHandler \(page 1160\)](#)
- | [setTextBoxHandler \(page 1170\)](#)
- | [setTimeBoxHandler \(page 1175\)](#)
- | [setWindowHandler \(page 1188\)](#)

setRangesFollowActiveClip

Description

Summary

Set play range to follow the active clip's play range.

Details

Set play range to follow the active clip's play range.

Functional area

Playback control

Command syntax

Syntax

```
setRangesFollowActiveClip boolVal
```

Arguments

Name	Type	Required	Comments
boolVal	integer	yes	True False Toggle

Flags

None

Return value

void

Examples

```
// Set play range to follow the active clips play Range.  
setRangesFollowActiveClip true;
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

setRotation

Description

Summary

Set the rotation on the specified modules.

Details

Use setRotation to set the rotation of a module in relative or world space coordinates.

setRotation modifies the keys of the current frame. If no animation data exists when setRotation is run, it creates a key.

Functional area

Data manipulators

Command syntax

Syntax

```
setRotation "moduleName" rotVec[-ws] [-preRot] [-def]
```

Arguments

Name	Type	Required	Comments
rotVec			The vector data to apply
moduleName			Name of the module you want to modify

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	0	—	—	RotVec data is applied in world space coordinates
preRot	0	—	def	—
def	0	—	preRot	—

Return value

void

Examples

```
//Create a marker and set its rotation
create Marker "R_UpperArm";
setRotation R_UpperArm <<100, 0, 0>>;
```

Additional information

Related commands

- | [setKey \(page 1083\)](#)
- | [setPosition \(page 1123\)](#)
- | [setScale \(page 1147\)](#)

setSavePath

Description

Summary

Set the default directory that Shogun Post's **Open** and **Save As** dialog boxes initialize to.

Details

Sets the default directory that Shogun Post's **Open** and **Save As** dialog boxes initialize to, or the default directory of the **Export** dialog if the `-e` flag is specified.

Shogun Post's **Open** and **Save** dialog boxes share their default values. However, the **Import** and **Export** dialog boxes have different default values. The values of these may be retrieved using the [getSavePath \(page 664\)](#) command. The values get updated whenever you use the dialog boxes.

Use forward slashes instead of back-slashes for all file and directory paths.

Functional area

System

Command syntax

Syntax

```
setSavePath "savePath" [-e]
```

Arguments

Name	Type	Required	Comments
savePath	string	yes	The path files should be saved to

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
e	0	—	—	—

Return value

void

Examples

```
// Make the export dialog default to the temp directory
setSavePath -e "C:/Temp/";
```

Additional information

Related commands

■ [getSavePath \(page 664\)](#)

setScale

Description

Summary

Set the scale on the specified modules.

Details

Use setScale to set the scale of a module in relative or world space coordinates.

setScale modifies the keys of the current frame. If no animation data exists when setScale is run, it creates a key.

Functional area

Data manipulators

Command syntax

Syntax

```
setScale "moduleName" scaleVec[-ws] [-def]
```

Arguments

Name	Type	Required	Comments
scaleVec			The vector data to apply
moduleName			Name of the module you want to modify

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ws	0	—	—	ScaleVec data is applied in world space coordinates
def	0	—	—	—

Return value

void

Examples

```
//Create a marker and set its scale
create Marker "marker1";
setScale marker1 <<100, 0, 0>>;
```

Additional information

Related commands

- | [setKey \(page 1083\)](#)
- | [setPosition \(page 1123\)](#)
- | [setRotation \(page 1143\)](#)

setSceneName

Description

Summary

Sets the name of the scene.

Details

Sets the name of the scene. The scene name gets set automatically when opening or importing a file, and is the default name that appears in the **Save**, **Save As**, and **Export** dialog boxes.

Functional area

System

Command syntax

Syntax

```
setSceneName "name"
```

Arguments

Name	Type	Required	Comments
sceneName	string	yes	The name of the scene.

Flags

None

Return value

void

Examples

```
// Modify the name of the scene
string $name;

// Append v2 to the current scene name
$name = `getSceneName`;
$name += "_v2";

// Set it
setSceneName $name;
```

Additional information

Related commands

■ [getSceneName \(page 668\)](#)

setScriptPaths

Description

Summary

Sets the list of script directories where Shogun Post will look for scripts and pipelines that can be executed.

Details

Sets the list of script directories where Shogun Post will look for scripts and pipelines that can be executed. The script directories are reparsed immediately. This means that if your script calls another script that isn't under any of the new directories, your script will fail after executing this command.

Functional area

System

Command syntax

Syntax

```
setScriptPaths ScriptPaths
```

Arguments

Name	Type	Required	Comments
ScriptPaths	string array	yes	String array of the new script directories.

Flags

None

Return value

void

Examples

```
// Set the script directories
string $scriptPaths\[4\];
$scriptPaths[0] = "C:/Program Files/Vicon/ShogunPost1.0/MyLiveScripts/";
$scriptPaths[1] = "C:/Program Files/Vicon/ShogunPost1.0/MyLabeling/";
$scriptPaths[2] = "C:/Program Files/Vicon/ShogunPost1.0/MyEditingScripts/";
$scriptPaths[3] = "C:/Program Files/Vicon/ShogunPost1.0/MySolvingScripts/";
setScriptPaths $scriptPaths;
```

Additional information

Related commands

- | [addScript \(page 121\)](#)
- | [addScriptPath \(page 123\)](#)
- | [clearScriptPaths \(page 208\)](#)
- | [deleteScriptPath \(page 346\)](#)
- | [getScriptPaths \(page 670\)](#)
- | [scriptExists \(page 952\)](#)

setSelectionFilter

Description

Summary

Toggles the on/off state of selection filters found in the 3D view selection filter drop down.

Details

One of the active views must be a 3D view for this command to work.

Functional area

Interface

Command syntax

Syntax

```
setSelectionFilter "objectName" boolean
```

Arguments

Name	Type	Required	Comments
objectName	string	yes	Name of the object or module type whose selection state is to be set.
boolean	boolean	yes	Boolean to turn on, off, or toggle the selection state of the named object or module.

Flags

None

Return value

void

Example

```
//Toggling the Marker selection filter  
setSelectionFilter "Marker" toggle;
```

Additional information

Related commands

- | [getSelectionFilter \(page 676\)](#)
- | [getViewFilter \(page 719\)](#)
- | [setViewFilter \(page 1187\)](#)

setSelectionSetFile

Description

Summary

Set the file where Shogun Post loads and saves your selection sets.

Functional area

System

Command syntax

Syntax

```
setSelectionSetFile SelectionSetFile[-save]
```

Arguments

Name	Type	Required	Comments
SelectionSetFile	string	yes	Name of the file where Shogun Post loads and saves your selection sets. Use forward-slashes instead of back-slashes for all file paths.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
save	0	—	—	Causes the current selection sets to be saved to the given file. If not specified, then the selection sets in the file will be loaded.

Return value

void

Examples

```
// Save your selection sets to the root of your C: drive  
setSelectionSetFile "C:/MySelectionSet.hsl" -save;
```

Additional information

Related commands

- | [setAutoSaveFile \(page 1027\)](#)
- | [setHotKeyFile \(page 1079\)](#)
- | [setLogFile \(page 1104\)](#)

setSession

Description

Summary

Sets the active session to the given session path.

Details

Replaces the `-sessionFolder` option of the deprecated `captureOptions` command.

Functional area

Data management

Command syntax

Syntax

```
setSession ["sessionPath"]
```

Return value

void

setSolversFirst

Description

Summary

Switches the button on and off.

Details

Switches the button on and off. Useful when blending and looping data.

Functional area

Interface

Command syntax

Syntax

```
setSolversFirst boolean
```

Arguments

Name	Type	Required	Comments
boolean	Boolean	yes	Switches the button on and off.

Flags

None

Return value

void

Examples

```
//Turn off the solvers after clip/blends  
setSolversFirst false;
```

Additional information

Related commands

- | [solve \(page 1226\)](#)

setStaticBoxHandler

Description

Summary

Sets the event handler for the user static box.

Details

Sets the event handler for the static box user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setStaticBoxHandler userControlID [-click string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
click	1	string	—	Specifies the script to call when the user clicks on the static box.

Return value

void

Examples

```
// Set the event handler for the User Static Box.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createStaticBox $windowId -text "Static Text"`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "StaticBoxId" $controlId;

// Set the -click event handler for the Static Box. When
// the user mouse clicks on the Static Box, this script will be
// executed. StaticBoxClickHandler must reside in one of
// the scripts directories.
setStaticBoxHandler $controlId -click "StaticBoxClickHandler";
```

Additional information

Related commands

- [setCheckBoxHandler \(page 1035\)](#)
- [setColorPickerHandler \(page 1040\)](#)
- [setDropListHandler \(page 1058\)](#)
- [setListBoxHandler \(page 1088\)](#)

-
- | [setNumBoxHandler \(page 1111\)](#)
 - | [setPushButtonHandler \(page 1133\)](#)
 - | [setRadioButtonHandler \(page 1138\)](#)
 - | [setTextBoxHandler \(page 1170\)](#)
 - | [setTimeBoxHandler \(page 1175\)](#)
 - | [setWindowHandler \(page 1188\)](#)

setStaticParameterExpression

Description

Summary

Sets the expression of a static parameter of the given name.

Details

Functional area

Parameters

Command syntax

Syntax

```
setStaticParameterExpression parameterName parameterValueOrExpression [-  
onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	yes	Specifies the name of the parameter added
parameterValueOrExpression			

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod		string	—	May be used to specify the character on which the parameter exists. Without this option, the selected character is used.
solving				Specifies if the parameter exists on the solving setup instead of the labeling setup

Return value

void

setStaticParameterUserValue

Description

Summary

Sets the user value of a static parameter of the given name.

Details

Functional area

Parameters

Command syntax

Syntax

```
setStaticParameterUserValue parameterName value [-onMod string] [-solving]
```

Arguments

Name	Type	Required	Comments
parameterName	string	yes	Specifies the name of the parameter added

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onMod		string	—	May be used to specify the character on which the parameter exists. Without this option, the selected character is used.
solving				Specifies if the parameter exists on the solving setup instead of the labeling setup

Return value

void

setTabHandler

Description

Summary

Set the event handler for a tab controller.

Details

Set the event handler for a tab controller specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setTabHandler userControlID [-selChange string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of the control to be queried.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
selChange	1	string	—	The script that the command will be referencing.

Return value

void

Examples

```
// Set the event handler for the User Num Box.
int $windowId;
int $controlId;

// Create a Tab controller and forms to associate with the tabs.
int $mainTab;
int $mainForm;
int $secondForm;
int $thirdForm;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

//create a main tab controller with a second and third tab
$mainTab = `createTab $windowId "Main" "Second" "Third" -sel 2`;

// Set the -selChange event handler for the setTabHandler. When
// the user switches tabs, this script will be
// executed. setTabHandler must reside in one of
// the scripts directories.
setTabHandler $mainTab -selChange " SetTabHandler";
```

Additional information

Related commands

- [addTab \(page 127\)](#)
- [createTab \(page 300\)](#)

-
- [deleteAllTabItems \(page 330\)](#)
 - [deleteTabItem \(page 352\)](#)
 - [findTabItem \(page 416\)](#)
 - [getTabItem \(page 697\)](#)
 - [getTabSellItem \(page 699\)](#)
 - [selectTabItem \(page 1007\)](#)

setTextBoxHandler

Description

Summary

Sets the event handler for the user text box.

Details

Sets the event handler for the text box user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setTextBoxHandler userControlID [-change string] [-loseFocus string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
change	1	string	—	Specifies the script to call when the user changes the value of the text box.
loseFocus	1	string	—	Specifies the script to call when the text box loses the input focus.

Return value

void

Examples

```
// Set the event handler for the User Text Box.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createTextBox $windowId`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "TextBoxId" $controlId;

// Set the -loseFocus event handler for the Text Box. When
// the user clicks (or Tabs) out of Text Box item, this script will be
// executed. TextBoxLoseFocusHandler must reside in one of
// the scripts directories.
setTextBoxHandler $controlId -loseFocus "TextBoxLoseFocusHandler";
```

Additional information

Related commands

- [setCheckBoxHandler \(page 1035\)](#)
- [setColorPickerHandler \(page 1040\)](#)
- [setDropListHandler \(page 1058\)](#)
- [setListBoxHandler \(page 1088\)](#)
- [setNumBoxHandler \(page 1111\)](#)
- [setPushButtonHandler \(page 1133\)](#)
- [setRadioButtonHandler \(page 1138\)](#)
- [setStaticBoxHandler \(page 1160\)](#)
- [setTimeBoxHandler \(page 1175\)](#)
- [setWindowHandler \(page 1188\)](#)

setTime

Description

Summary

Set current frame number. No frame sets time to beginning of play range.

Details

Use setTime to explicitly set the current frame of the animation.

No argument sets time to the beginning of the play range.

SMPTE times must be surrounded by quotes.

Functional area

Playback control

Command syntax

Syntax

```
setTime timeVal
```

Arguments

Name	Type	Required	Comments
frameNumber	integer	yes	Float value representing frame number or time

Flags

None

Return value

void

Examples

```
setTime -s 2;  
// Assuming a 60 fps scene, this command will set the current  
// time to frame 120.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

setTimeBoxHandler

Description

Summary

Sets the event handler for the user time box.

Details

Sets the event handler for the time box user control specified by `userControlID`.

Event handlers are simply scripts to be called when certain events happen. User controls get notified about changes in state when users interact with them. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user control will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setTimeBoxHandler userControlID [-change string] [-loseFocus string]
```

Arguments

Name	Type	Required	Comments
<code>userControlId</code>	int	yes	ID of user control to set event handler for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
change	1	string	—	Specifies the script to call when the user changes the value of the time box.
loseFocus	1	string	—	Specifies the script to call when the text box loses the input focus.

Return value

void

Examples

```
// Set the event handler for the User Time Box.
int $windowId;
int $controlId;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createTimeBox $windowId`;

// Save the control ID to our profile, so we can retrieve it
// from the event handler
writeProfileInt "MyWindowSection" "TimeBoxId" $controlId;
// Set the -loseFocus event handler for the Time Box. When
// the user clicks (or Tabs) out of Time Box item, this script will be
// executed. TimeBoxLoseFocusHandler must reside in one of
// the scripts directories.
setTimeBoxHandler $controlId -loseFocus "TimeBoxLoseFocusHandler";
```

Additional information

Related commands

- [setCheckBoxHandler \(page 1035\)](#)
- [setColorPickerHandler \(page 1040\)](#)

-
- [setDropListHandler \(page 1058\)](#)
 - [setListBoxHandler \(page 1088\)](#)
 - [setNumBoxHandler \(page 1111\)](#)
 - [setPushButtonHandler \(page 1133\)](#)
 - [setRadioButtonHandler \(page 1138\)](#)
 - [setStaticBoxHandler \(page 1160\)](#)
 - [setTextBoxHandler \(page 1170\)](#)
 - [setWindowHandler \(page 1188\)](#)

setTimeBoxTime

Description

Summary

Set the user time box time value.

Details

Sets the time value of the time box user control specified by `userControlID`. The time value should be specified in frames.

Functional area

User Window

Command syntax

Syntax

```
setTimeBoxTime userControlID frame
```

Arguments

Name	Type	Required	Comments
<code>timeVal</code>	int	yes	Time value, in frames.
<code>userControlId</code>	int	yes	ID of user control to operate on.

Flags

None

Return value

void

Examples

```
// Get the time value of a Time Box User Control
int $windowId;
int $controlId;
int $time;

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Create the User Control on the Window.
$controlId = `createTimeBox $windowId`;

// Set the time to some arbitrary frame
setTimeBoxTime $controlId 125;

// Get the time we just set
$time = `getTimeBoxTime $controlId`;
print $time;
```

Additional information

Related commands

- | [createTimeBox \(page 308\)](#)
- | [getTimeBoxTime \(page 703\)](#)
- | [setTimeBoxHandler \(page 1175\)](#)

setTimeCodeOffset

Description

Summary

Specify a time code offset to imported data in order to sync that data with the data in the current scene

Details

Set the SMPTE offset (in frames) on a clip. If no clip name is specified, then the offset will be applied to the active clip.

Functional area

Data editing

Command syntax

Syntax

```
setTimeCodeOffset offset [-onClip string]
```

Arguments

Name	Type	Required	Comments
offset	string	yes	The offset to apply

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
onClip	1	string	—	Name of the clip on which to set the time code offset

Return value

void

Examples

```
// Set the SMPTE offset on the active clip
int $offset = 4;
setTimeCodeOffset $offset;
```

Additional information

Related commands

- [getTimeCodeOffset \(page 706\)](#)

setTimeFormat

Description

Summary

Specifies whether various parts of the UI display time in SMPTE timecode format or frame number. Most notably this is used for certain time attributes.

Details

Functional area

System

Command syntax

Syntax

```
setTimeFormat "smpteOrFrames"
```

Arguments

Name	Type	Required	Comments
smpteOrFrames	string	yes	Must be either " smpte" or "frames"

Flags

None

Return value

void

setTrackList

Description

Summary

Set the list of objects that the 3D views should track/follow.

Details

Sets the list of objects that the 3D views should track/follow for the cameraView "Track Objects" tracking mode.

"Track Objects" forces the camera to follow a set of objects. Use this command to set or modify the list of objects that the camera will follow. It uses the selected objects in the scene to set the list. The "Track Objects" tracking mode is helpful when editing multi-person shots, or when following a group of objects but changing selection frequently.

Functional area

Interface

Command syntax

Syntax

```
setTrackList [-a] [-r]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
a	0	—	r	Add to the list of objects.
r	0	—	a	Remove from the list of objects.

Return value

void

Examples

```
// Select a set of markers to tracks
selectSet "Waist";

// Set the tracking mode on the 3D views
cameraView -track "Objects";

// Set the list of objects to track
.setTrackList;
```

Additional information

Related commands

■ [cameraView \(page 196\)](#)

setUseAxiomSolving

Description

Summary

Sets the status of the "Axiom Solving" flag.

Details

Sets whether the Axiom solving algorithm will be used by the solve command. If true, Axiom solving is used; if false, the legacy solver is used instead.

Functional area

Skeletal solving

Command syntax

Syntax

```
setUseAxiomSolving
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

- [getUseAxiomSolving \(page 715\)](#)

setViewFilter

Description

Summary

Sets view filter.

Functional area

Interface

Command syntax

Syntax

```
setViewFilter "ObjectName" bool(on/off/toggle)
```

Arguments

See above syntax.

Flags

None

Return value

void

Additional information

Related commands

■ [getViewFilter \(page 719\)](#)

setWindowHandler

Description

Summary

Sets the event handlers for the user window.

Details

Sets the event handlers for the user window.

Event handlers are simply scripts to be called when certain events happen. User windows get notified about changes in the scene as well as changes to the user window itself. Each event handler flag takes as an argument a script (which must reside in your script directories).

To clear an event handler, pass the empty string, "", to the flag. If no options are specified, all handlers are reset, and the user window will not respond to any events.

Functional area

User Window

Command syntax

Syntax

```
setWindowHandler userWindowID [-enter string] [-size string] [-dataChange  
string] [-attChange string] [-selChange string] [-timeChange string] [-  
newScene string] [-fileLoaded string] [-markerLabeled string] [-  
markerUnlabeled string]
```

Arguments

Name	Type	Required	Comments
windowId	int	yes	ID of user window to set event handlers for.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
enter	1	string	—	Specifies the script to call when the user has pressed the Enter key while a user control has the input focus.
size	1	string	—	Specifies the script to call when the user window's size has changed due to manual resize, or docking/undocking.
dataChange	1	string	—	Specifies the script to call when data has changed in the scene. This can include animation changes, hierarchy changes, new/deleted modules, etc.
attChange	1	string	—	Specifies the script to call when a module's attribute has changed.
selChange	1	string	—	Specifies the script to call when the selection in the scene changes.
timeChange	1	string	—	Specifies the script to call when either the current time, play range, or animation range has changed in the scene.
newScene	1	string	—	Specifies the script to call when the scene is reset.
fileLoaded	1	string	—	—
markerLabeled	1	string	—	—
markerUnlabeled	1	string	—	—

Return value

void

Examples

```
// Set the event handler for a User Window. We will set
// the "enter" event handler.
int $windowId;

// First create a User Window to place the Controls on
$windowId = `createWindow "MyWindow"`;

// Save the window ID to our profile, so we can retrieve it
// from the event handlers
writeProfileInt "MyWindowSection" "WindowId" $windowId;

// Set the -enter event handler for the User Window. When
// the user hits the enter button, this script will be
// executed. MyScriptEnterHandler must reside in one of
// the scripts directories.
setWindowHandler $windowId -enter "MyScriptEnterHandler";
```

Additional information

Related commands

- | [setCheckBoxHandler \(page 1035\)](#)
- | [setColorPickerHandler \(page 1040\)](#)
- | [setDropListHandler \(page 1058\)](#)
- | [setListBoxHandler \(page 1088\)](#)
- | [setNumBoxHandler \(page 1111\)](#)
- | [setPushButtonHandler \(page 1133\)](#)
- | [setRadioButtonHandler \(page 1138\)](#)
- | [setStaticBoxHandler \(page 1160\)](#)
- | [setTextBoxHandler \(page 1170\)](#)
- | [setTimeBoxHandler \(page 1175\)](#)

setWindowSize

Description

Summary

Set the size of a docking window (or pane)

Details

Enables you to set the size of a docking window (or pane). You must specify either the `-width` or `-height` flag, or both, otherwise the command has no effect.

Note that sizing docked panes is not yet supported (only floating panes).

Functional area

Interface

Command syntax

Syntax

```
setWindowSize "windowName" [-width integer] [-height integer]
```

Arguments

Name	Type	Required	Comments
windowName	string	true	The name of the docking window (or pane) to size

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
width	1	integer	—	The new width of the window, in pixels
height	1	integer	—	The new height of the window, in pixels

Return value

void

Examples

```
// This sets the width of the "Attributes" window to 200 pixels
setWindowSize "Attributes" -width 200;
```

Additional information

Related commands

- | [autohideWindow \(page 156\)](#)
- | [dockWindow \(page 361\)](#)
- | [floatWindow \(page 428\)](#)
- | [showWindow \(page 1197\)](#)
- | [tabWindow \(page 1306\)](#)

show

Description

Summary

Turn on the visibility of modules in the scene.

Details

Use show when you want to turn on the visibility of named objects in your scene. No arguments turns on visibility for all modules.

Functional area

Data manipulators

Command syntax

Syntax

```
show name1 name2 ...[-s]
```

Arguments

Name	Type	Required	Comments
name			Name of modules in the scene.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
s	0	—	—	Show only the selected objects

Return value

void

Examples

```
show RANK; // Sets visibility of RANK module to "on".  
show; // Set visibility of all modules to "on"
```

Additional information

Related commands

■ [hide \(page 741\)](#)

showControl

Description

Summary

Show/hide a user control.

Details

Shows or hides the user control specified by `userControlID`. All user controls are visible by default, unless created with the `-hidden` option.

Functional area

User Window

Command syntax

Syntax

```
showControl userControlID showBoolean
```

Arguments

Name	Type	Required	Comments
<code>showBoolean</code>	boolean	yes	Flag to show/hide the user control
<code>userControlId</code>	int	yes	ID of user control to show/hide.

Flags

None

Return value

boolean

Examples

```
// Show/Hide a control.
int $windowId;
int $controlId;
boolean $wasShowing;

// First create a User Window to place the Control on
$windowId = `createWindow "MyWindow"`;

// Create a Push Button Control in the window. Create it hidden.
// We will show it later.
$controlId = `createPushButton $windowId -text "Sample" -hidden`;

// Show the Control, getting its previous visibility
$wasShowing = `showControl $controlId true`;
print $wasShowing;
```

Additional information

Related commands

- | [enableControl \(page 365\)](#)
- | [getControlEnabled \(page 489\)](#)
- | [getControlPos \(page 491\)](#)
- | [getControlText \(page 493\)](#)
- | [getControlVisibility \(page 495\)](#)
- | [getFocus \(page 545\)](#)
- | [setControlPos \(page 1048\)](#)
- | [setControlText \(page 1051\)](#)
- | [setControlTip \(page 1053\)](#)
- | [setFocus \(page 1070\)](#)

showWindow

Description

Summary

Set visibility of a docking window.

Details

Sets the visibility of a docking window. The docking window can be either a Shogun Post panel or a user window.

Functional area

User Window

Command syntax

Syntax

```
showWindow "windowName" showBoolean
```

Arguments

Name	Type	Required	Comments
windowName	string	yes	Name of the window to set visibility for. Can be one of a Shogun Post panel or a user window.
boolVal	boolean	yes	Boolean to show, hide, or toggle visibility of the docking window.

Flags

None

Return value

void

Examples

```
// Toggle the visibility of the Attributes panel  
showWindow "Attributes" toggle;
```

Additional information

Related commands

■ [getWindowVisibility \(page 725\)](#)

sin

Description

Summary

Returns the sine of an input angle (degrees)

Details

The sin command is used to calculate the sine for any input angle. The input units are in degrees.

Picture the unit circle (a circle of radius 1.0 centered about the origin). If you were to imagine a unit vector rotating inside of the unit circle, the angle (in degrees) that the unit vector makes with the positive x-axis is the input argument theta. The y-coordinate of the endpoint of the unit vector is the return value of the sin function.

Functional area

Math

Command syntax

Syntax

```
sin value
```

Arguments

Name	Type	Required	Comments
value	float	yes	Float value of an angle in degrees.

Flags

None

Return value

float

Returns a floating point value in the range [-1.0, 1.0]

Examples

```
float $temp = (sin (30));  
// declares a variable for the sine function  
  
print (string ($temp));  
// prints the output of the sine function
```

Additional information

Related commands

- | [acos \(page 103\)](#)
- | [asin \(page 144\)](#)
- | [atan \(page 148\)](#)
- | [cos \(page 230\)](#)
- | [tan \(page 1308\)](#)

skelSetup

Description

Summary

Automatically attaches markers to bones based on distance.

Details

Assumes a skeleton is posed in a marker cloud. Creates a constraint for each marker and sets that constraint to act on the closest bone to the marker in the scene.

Can be used to automate skeleton setup, but resulting constraints should be reviewed as the closest bone may not be the bone you want a marker to constrain.

Functional area

Skeletal solving

Command syntax

Syntax

```
skelSetup
```

Arguments

None

Flags

None

Return value

void

Examples

```
//Run this with a skeleton posed in a marker cloud and no  
//existing constraints.  
skelSetup;
```

Additional information

Related commands

- | [attach \(page 150\)](#)
- | [setConstraint \(page 1043\)](#)
- | [solve \(page 1226\)](#)

slave

Description

Summary

Turn the DBS slave on or off

Details

This command allows you to enable or disable a DBS (Distributed Batch System) slave.

A slave seat provides processing services to the DBS master seat, receiving tasks from the master seat and returning the results. A DBS slave can only actively process requests from one master at a time. In order for the master and slave to communicate, they must be on the same workgroup and port.

Functional area

Master/Slave

Command syntax

Syntax

```
slave on/off/toggle [-port integer] [-workgroup string]
```

Arguments

Name	Type	Required	Comments
on/off/toggle	boolean	yes	Turn the slave on or off

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
port		integer		Sets the port that the slave uses to listen for broadcasts from the master. It accepts a number value from 1 to 49150 and it must be set to the same port number as the master.
workgroup	1	string	—	Sets this slave as part of the given processing workgroup. See details for discussion about processing.

Return value

void

Examples

```
// This command will enable a DBS slave and join it as part of
// the "Farming" processing group.
slave on -workgroup "Farming";
slave toggle -workgroup "DBSWorkgroup" - port 5419;
```

Additional information

Related commands

■ [master \(page 794\)](#)

smpteToFrame

Description

Summary

Convert a SMPTE time code into a frame number

Details

Converts a SMPTE timecode string into a frame number, taking into account the SMPTE_Offset value of the active clip.

The SMPTE_Offset value of the active clip specifies the timecode start for the data on the clip. It is there so that frame 1 of your data corresponds to the starting timecode of your take. Without it, your frame numbers would end up large and cumbersome.

If you want to ignore the SMPTE_Offset in the computation, specify the `-ignoreOffset` flag. You may wish to do this when computing a SMPTE_Offset of your own, for instance. In that situation, you wouldn't want the existing offset value in the computation.

The frames (FF) portion of the timecode string should go from 0 to 24, 25, or 30 depending on the current timecode standard (Film, PAL, and NTSC respectively). As your scene rate may be at a frame rate that is a higher multiple of those standard rates (e.g. 120 fps for NTSC), the sub-frame portion indicates the "in between frame" of the frame passed in.

Functional area

System

Command syntax

Syntax

```
smpteToFrame "smpteStr" [-ignoreOffset]
```

Arguments

Name	Type	Required	Comments
smpteStr	string	yes	The timecode value to convert to a frame number.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ignoreOffset	0	—	—	By default, the active clip's SMPTE_Offset value is removed from the timecode argument to produce a final frame value. This flag causes the offset to be ignored causing a straight timecode to frames conversion.

Return value

integer

Examples

```
// Set the timecode offset of the active clip
setProperty "SMPTE_Offset" "01:23:45:12(2)" -onMod "TheActiveClip";

// Now convert a the value at exactly the one hour mark into a frame
// number.
int $frameNumber = `smpteToFrame "01:00:00:00(0)"`;
print $frameNumber;
```

Additional information

Related commands

- [frameToSmppte \(page 432\)](#)
- [getTime \(page 701\)](#)

snapTo

Description

Summary

Snaps the primary node to the average (translation and rotation) of the other selected nodes.

Details

Use snapTo when you need to position or orient an object based on the position and/or rotation of one or more other objects, whether at a single frame in the timeline, over a specific subset of the timeline, or over the entire animation range.

Functional area

Snap

Command syntax

Syntax

```
snapTo [-allTime] [-ranges] [-positionOnly] [-rotationOnly] [-system]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all time	0	—	ranges	Performs the snap over the entire animation range.
ranges	0	—	allTime	Performs the snap over all selected time ranges.

Name	Flag arguments	Argument type	Exclusive to	Comments
positionOnly	0	—	rotationOnly	Only modifies the translation of the primary node.
rotationOnly	0	—	positionOnly	Only modifies the rotation of the primary node.
system	0	—	positionOnly	Works similarly to snapToSystem (page 1220) command, but it doesn't position the markers according to the system, it only orients them to it. (Must be at least 4 markers selected for this option).

Return value

void

Examples

```
// This sequence of commands creates a marker called "rootTemp"
// and snap its position over all time to the average of the four
// indicated waist markers.
create Marker rootTemp;
select LFWT LBWT RFWT RBWT rootTemp;
snapTo -allTime;
```

Additional information

Related commands

- | [copyPattern \(page 226\)](#)
- | [snapToConstraint \(page 1209\)](#)
- | [snapToLine \(page 1212\)](#)
- | [snapToLocal \(page 1214\)](#)
- | [snapToRigid \(page 1217\)](#)
- | [snapToSystem \(page 1220\)](#)
- | [snapToSystemAlign \(page 1223\)](#)

snapToConstraint

Description

Summary

Snap selected markers to their constraint offset location.

Details

Sets the location of the selected markers to the offset of their constraint based on the current pose of the skeleton.

Useful during skeleton setup, when manually creating a VST, and as a method of filling gaps.

Note that [fillGaps \(page 389\)](#) -constraints may be more appropriate for gap filling. If no flags are used, the marker(s) only snap into place for the current frame.

Functional area

Snap

Command syntax

Syntax

```
snapToConstraint [-ranges] [-allTime] [-defaultValue] [-solvingOnly] [-labelingOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	0	—	allTime, defaultValue	Snaps the marker(s) only over the selected ranges
all time	0	—	ranges, defaultValue	Snaps the marker(s) over all time
defaultValue	0	—	ranges, allTime	Snaps the marker(s) at the current frame and only modifies their default values.
solvingOnly				
labelingOnly				

Return value

void

Examples

```
// Select a marker that is the source of a constraint. Move the marker
// away from its constraint offset location. Execute snapToConstraint
// to move it back to its constraint offset location.
snapToConstraint;
```

Additional information

Related commands

- | [createSkelScript \(page 291\)](#)
- | [setConstraint \(page 1043\)](#)
- | [snapTo \(page 1207\)](#)
- | [snapToLine \(page 1212\)](#)
- | [snapToLocal \(page 1214\)](#)
- | [snapToRigid \(page 1217\)](#)
- | [snapToSystem \(page 1220\)](#)

-
- [snapToSystemAlign \(page 1223\)](#)
 - [solve \(page 1226\)](#)

snapToLine

Description

Summary

Positions the given node along the line formed by two selected nodes.

Details

With two nodes that define a line select, snapToLine positions the targetNodeName along that line. Where along the line is specified as a percentage of the distance between the two nodes making up the line unless the `-absolute` option is used. When `-absolute` is used the distance along the line is specified in mm.

Functional area

Snap

Command syntax

Syntax

```
snapToLine "targetNodeName" %s [-allTime] [-absolute] [-ranges]
```

Arguments

Name	Type	Required	Comments
targetNodeName	string	yes	Specifies the name of the node to be snapped.
%s	float	yes	Specifies distance along the line the node should be snapped to.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
allTime	0	—	ranges	Performs the snap over the entire animation range.
absolute	0	—	—	Specifies the argument is in mm rather than a percentage of the distance between the two nodes that define the line.
ranges	0	—	allTime	Performs the snap over the selected time ranges.

Return value

void

Examples

```
// Snap LMWT half way between LFWT and LBWT at the current frame only.
select LFWT LBWT;
snapToLine LMWT 0.5;
```

Additional information

Related commands

- [snapTo \(page 1207\)](#)
- [snapToConstraint \(page 1209\)](#)
- [snapToLocal \(page 1214\)](#)
- [snapToRigid \(page 1217\)](#)
- [snapToSystem \(page 1220\)](#)
- [snapToSystemAlign \(page 1223\)](#)

snapToLocal

Description

Summary

Snaps named node to position and rotation defined by the local coordinate system of the selected node.

Details

The snapToLocal command differs from [snapToSystem \(page 1220\)](#), [snapToSystemAlign \(page 1223\)](#), and [snapToRigid \(page 1217\)](#) in that snapToLocal only requires one node be selected to define the coordinate system.

Functional area

Snap

Command syntax

Syntax

```
snapToLocal "targetNodeName" offsetX offsetY offsetZ [-allTime] [-positionOnly] [-rotationOnly] [-ranges]
```

Arguments

Name	Type	Required	Comments
zOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Z axis of the space defined by the selected markers.
yOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Y axis of the space defined by the selected markers.

Name	Type	Required	Comments
xOffset	float	yes	Specifies the position offset to apply to the object being snapped on the X axis of the space defined by the selected markers.
targetNodeName	string	yes	Name of the node that will be snapped.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all time	0	—	ranges	Performs the snap over the entire animation range.
positionOnly	0	—	rotationOnly	Only modifies the position of the node being snapped.
rotationOnly	0	—	positionOnly	Only modifies the rotation of the node being snapped.
absolute	0	—	—	Specifies the offset is in mm rather than a percentage of the distance between the first two nodes that define the system.
ranges	0	—	allTime	Performs the snap over the selected time ranges.

Return value

void

Examples

```
// Create a marker and set its translation and rotation in space
// then create a second marker and snap it relative to the first
// markers system.
create Marker "marker1";
setProperty Translation 200 0 200 -d ;
setProperty Rotation 0 45 45 -d;
```

```
create Marker "marker2";  
select marker1;  
snapToLocal "marker2" 100 100 100;
```

Additional information

Related commands

- | [snapTo \(page 1207\)](#)
- | [snapToConstraint \(page 1209\)](#)
- | [snapToLine \(page 1212\)](#)
- | [snapToRigid \(page 1217\)](#)
- | [snapToSystem \(page 1220\)](#)
- | [snapToSystemAlign \(page 1223\)](#)

snapToRigid

Description

Summary

Snaps named node to a position and/or rotation formed by a rigid body defined from the currently selected nodes.

Details

Use snapToRigid when you need to constrain a node to a location defined by a rigid body operation.

If neither option for `allTime` or `ranges` is used, then the operation will affect the current frame only.

The units for the `-absolute` option are millimeters.

Values for the default `relative` option are a percentage of the distance from the first selected node to the second selected node.

Use of the `-positionOnly` option prevents rigid body rotations from being applied to the target node. This option can be useful when you want to snap an object, like a baseball bat to a new location while maintaining the rotational orientation of the bat.

snapToRigid is different from [snapToSystem \(page 1220\)](#) and [snapToSystemAlign \(page 1223\)](#) in that snapToRigid defines a rigid body out of the selected markers and uses its coordinate system. The rigid body is defined at a frame the command determines. snapToSystem and snapToSystemAlign do not use a rigid body, they refine the coordinate system at each frame.

Functional area

Snap

Command syntax

Syntax

```
snapToRigid "targetNodeName" xOffset yOffset zOffset [-allTime] [-ranges]
[-absolute] [-positionOnly]
```

Arguments

Name	Type	Required	Comments
zOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Z axis of the space defined by the selected markers.
yOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Y axis of the space defined by the selected markers.
xOffset	float	yes	Specifies the position offset to apply to the object being snapped on the X axis of the space defined by the selected markers.
targetNodeName	string	yes	Name of the node that will be snapped.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all time	0	—	ranges	Performs the snap over the entire animation range.
positionOnly	0	—	rotationOnly	Only modifies the position of the node being snapped.
rotationOnly	0	—	positionOnly	Only modifies the rotation of the node being snapped.
absolute	0	—	—	Specifies the offset is in mm rather than a percentage of the distance between the first two nodes that define the system.
ranges	0	—	allTime	Performs the snap over the selected time ranges.

Return value

void

Examples

```
// This sequence of commands will create a new Marker node
// and snap its position over all time to a position offset by
// the indicated number of millimeters ( -50 in X, 50 in Y,
// and 50 in Z) calculated from the origin of a temporary rigid
// body created from LFWT, LMWT, and LBWT. The offset is applied
// in the local coordinate system of the temporary rigid body.
// The rigid body coordinate system is then expressed in rotations
// relative to world space and applied to the target node,
// snapRigidTest.
create Marker snapRigidTest;
select LFWT LMWT LBWT;
snapToRigid -allTime -absolute snapRigidTest -50 50 50;
```

Additional information

Related commands

- | [snapTo \(page 1207\)](#)
- | [snapToConstraint \(page 1209\)](#)
- | [snapToLine \(page 1212\)](#)
- | [snapToLocal \(page 1214\)](#)
- | [snapToSystem \(page 1220\)](#)
- | [snapToSystemAlign \(page 1223\)](#)

snapToSystem

Description

Summary

Snaps named node to position and/or rotation defined by a coordinate system formed by three currently selected nodes.

Details

The snapToSystem command positions and/or orients a module based on a coordinate system defined by the positions of three selected nodes:

- The first selected node is always the origin
- The second selected node defines the X axis
- The third selected node defines the Y axis
- Z is normal to the plane XY.

This command is similar to [snapToRigid \(page 1217\)](#) with some important differences:

- snapToSystem requires exactly three nodes, snapToRigid may use three or more nodes.
- snapToSystem defines its coordinate system at each frame, whereas snapToRigid uses the entire frame range to establish the best coordinate system given the available data.

Use snapToSystem when you need to constrain a module to a location defined by a local coordinate system that is re-defined for each frame. If neither option for all time or ranges is used, then the operation affects the current frame only.

By default the offsets are defined as a percentage of the distance from the first selected module to the second selected module. Alternatively, the `-absolute` option can be used to define the offset in millimeters.

Use of the `-positionOnly` option prevents rigid body rotations from being applied to the target module. This option can be useful when you want to snap an object, like a baseball bat to a new location while maintaining the rotational orientation of the bat. If you want to aim an axis other than X of the module being snapped at the X axis, see [snapToSystemAlign \(page 1223\)](#).

Functional area

Snap

Command syntax

Syntax

```
snapToSystem "targetNodeName" xOffset yOffset zOffset [-allTime] [-positionOnly] [-rotationOnly] [-absolute] [-ranges]
```

Arguments

Name	Type	Required	Comments
zOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Z axis of the space defined by the selected markers.
yOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Y axis of the space defined by the selected markers.
xOffset	float	yes	Specifies the position offset to apply to the object being snapped on the X axis of the space defined by the selected markers.
targetNodeName	string	yes	Name of the node that will be snapped.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
allTime	0	—	ranges	Performs the snap over the entire animation range.
positionOnly	0	—	rotationOnly	Only modifies the position of the module being snapped.
rotationOnly	0	—	positionOnly	Only modifies the rotation of the module being snapped.

Name	Flag arguments	Argument type	Exclusive to	Comments
absolute	0	—	—	Specifies the offset is in mm rather than a percentage of the distance between the first two modules that define the system.
ranges	0	—	allTime	Performs the snap over the selected time ranges.

Return value

void

Examples

```
// This sequence of commands creates a new Marker module
// and snap its position over all time to a position offset the
// indicated number of millimeters ( -50 in X, 50 in Y, and 50 in Z)
// calculated from the origin of a coordinate system defined by the
// animation of LFWT, LMWT, and LBWT. The offset is applied in terms
// of the coordinate system defined at each frame.
// The coordinate system is then expressed in rotations relative to
// world space and applied to the target module, snapSystemTest.
create Marker snapSystemTest;
select LFWT LMWT LBWT;
snapToSystem -allTime -absolute snapSystemTest -50 50 50;
```

Additional information

Related commands

- [snapTo \(page 1207\)](#)
- [snapToConstraint \(page 1209\)](#)
- [snapToLocal \(page 1214\)](#)
- [snapToRigid \(page 1217\)](#)
- [snapToSystemAlign \(page 1223\)](#)

snapToSystemAlign

Description

Summary

Snaps a node based on the coordinate system formed by three selected markers with options for how the snapped node will be aligned.

Details

snapToSystemAlign is a more powerful version of snapToSystem.

snapToSystem aligns the X axis of the object being snapped at the second node selected and the Y axis of at the third node selected. This is often not the alignment you want. snapToSystemAlign allows you to specify which axis should point at the second node selected, and which axis should point at the third node selected. See [snapToSystem \(page 1220\)](#) for more information.

Functional area

Data editing

Command syntax

Syntax

```
snapToSystemAlign "targetNodeName" xOffset yOffset zOffset aimAxis upAxis  
[-allTime] [-positionOnly] [-rotationOnly] [-absolute] [-ranges]
```

Arguments

Name	Type	Required	Comments
upAxis	string	yes	Specifies which axis of the object being snapped should point to the third elected node which defines the Y axis of the local system.
aimAxis	string	yes	Specifies which axis of the object being snapped should point to the second selected node which defines the X axis of the local system.
zOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Z axis of the space defined by the selected markers.
yOffset	float	yes	Specifies the position offset to apply to the object being snapped on the Y axis of the space defined by the selected markers.
xOffset	float	yes	Specifies the position offset to apply to the object being snapped on the X axis of the space defined by the selected markers.
targetNodeName	string	yes	Name of the node that will be snapped.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
allTime	0	—	ranges	Performs the snap over the entire animation range.
positionOnly	0	—	rotationOnly	Only modifies the position of the node being snapped.
rotationOnly	0	—	positionOnly	Only modifies the rotation of the node being snapped.

Name	Flag arguments	Argument type	Exclusive to	Comments
absolute	0	—	—	Specifies the offset is in mm rather than a percentage of the distance between the first two nodes that define the system.
ranges	0	—	allTime	Performs the snap over the selected time ranges.

Return value

void

Examples

```
// Select 3 markers that define a coordinate system. First marker is
// the origin.
// Second marker is the X axis. Third marker is the Y axis.
select VirtualElbowMarker VirtualWristMarker LOWR;

// Snap and align a bone named LeftElbow so that it starts at the virtual
// elbow marker, aims its Z axis at the virtual wrist marker, and uses
// the left outer wrist marker as the up axis defining the twist along
// the Z axis.
snapToSystemAlign LeftElbow 0 0 0 "Z" "X" -allTime;
```

Additional information

Related commands

- [snapTo \(page 1207\)](#)
- [snapToConstraint \(page 1209\)](#)
- [snapToLocal \(page 1214\)](#)
- [snapToRigid \(page 1217\)](#)
- [snapToSystem \(page 1220\)](#)
- [snapToSystem \(page 1220\)](#)

solve

Description

Summary

Applies the currently active solvers to every frame in the current play range.

Details

The solve command executes the calculations that apply animation to a skeleton's bones based upon the constraints that pertain to them. Typically the constraints are between markers and bones. Any active solvers defined will be solved when this command is used. If the range option is not used, the command will solve the entire play range.

Functional area

Skeletal solving

Command syntax

Syntax

```
solve [-ranges] [-currentFrame] [-labelingSetup string] [-selectedCharacters]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
ranges	0	—	—	Solves only during the selected time ranges.
currentFrame				
labelingSetup		string		
selectedCharacters				

Return value

void

Examples

```
// This command executes calculations for the active constraints
// on the Bone Nodes that belong to the currently active solvers for
// the current selected time range. Note: not all of the solvers will
// necessarily be active, and not all of the constraints applied to
// the Bone Nodes have to be active.
solve -ranges;
```

Additional information

Related commands

■ [solver \(page 1228\)](#)

solver

Description

Summary

Performs operations on the named solver.

Details

A solver is a collection of nodes (typically bones) designed to be solved based on the constraints that exist on those nodes.

Using the solver commands, you may add or remove bones from the influence of the solver, you may add or remove constraints from the solver, and you may query the solver for the bones and constraint objects that belong to it.

To initially create a solver, see the [create \(page 234\)](#) command. All of the solver operations here are also available in the **Processing** panel.

Functional area

Skeletal solving

Command syntax

Syntax

```
solver "Solver" [-listBones] [-addBones] [-removeBones] [-removeAllBones]
[-selectBones] [-selectMarkers] [-selectConstraints] [-reset] [-cutKeys]
```

Arguments

Name	Type	Required	Comments
solverName	String	Yes	Specifies the name of the solver command acts on.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>listBones</code>	0	—	—	Prints the names of all bones associated with the given solver to the command log.
<code>addBones</code>	0	—	<code>removeBones</code> , <code>removeAllBones</code>	Adds all selected bones to the solver.
<code>removeBones</code>	0	—	<code>addBones</code> , <code>removeAllBones</code>	Removes all selected bones from the solver
<code>removeAllBones</code>	0	—	<code>addBones</code> , <code>removeBones</code>	Removes all bones from the named solver.
<code>selectBones</code>	0	—	—	Selects all bones associated with the given solver.
<code>selectMarkers</code>	0	—	—	Selects all markers that are associated with the solver
<code>selectConstraints</code>				
<code>reset</code>	0	—	—	Forces the solver to update its internal representation of the skeleton and constraints.
<code>cutKeys</code>				

Return value

void

Examples

```
// This command sequence will select existing BoneNodes
// and add them to the existing solver module called "right_leg".
select lfemur ltibia lfoot ltoe;
solver right_leg -addBones;
```

Additional information

Related commands

- | [create \(page 234\)](#)
- | [solve \(page 1226\)](#)

sortFloats

Description

Summary

Sort a list of floating point numbers stored in an array.

Details

Sort a list of floating point numbers stored in an array.

Functional area

System

Command syntax

Syntax

```
sortFloats floatArray [-d]
```

Arguments

Name	Type	Required	Comments
floatArray	float array	yes	The array to sort

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
d	0	—	—	Sorts in descending order. Default is to sort in ascending order.

Return value

float array

Returns the sorted array.

Examples

```
// Demonstrate a simple sort.  
float $array[3];  
$array[0] = 1.0;  
$array[1] = -1.5;  
$array[2] = 1e-6;  
float $sortedArray[] = `sortFloats $array`;  
print $sortedArray;
```

Additional information

Related commands

- | [sortInts \(page 1233\)](#)
- | [sortStrings \(page 1235\)](#)

sortInts

Description

Summary

Sort a list of integers point numbers stored in an array.

Details

Sort a list of integers numbers stored in an array.

Functional area

System

Command syntax

Syntax

```
sortInts intArray [-d]
```

Arguments

Name	Type	Required	Comments
intArray	integer array	yes	The array to sort

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
d	0	—	—	Sorts in descending order. Default is to sort in ascending order.

Return value

integer array

Returns the sorted array.

Examples

```
// Demonstrate a simple sort.  
int $array[3];  
$array[0] = 3;  
$array[1] = -1;  
$array[2] = 10;  
int $sortedArray[] = `sortInts $array`;  
print $sortedArray;
```

Additional information

Related commands

- | [sortFloats \(page 1231\)](#)
- | [sortStrings \(page 1235\)](#)

sortStrings

Description

Summary

Alphabetically sort a list of strings stored in an array.

Details

Alphabetically sort a list of strings stored in an array. The sorting will be case sensitive, meaning that the letters A-Z will get sorted before a-z.

Functional area

System

Command syntax

Syntax

```
sortStrings stringArray [-d] [-numeric]
```

Arguments

Name	Type	Required	Comments
stringArray	string array	yes	The array to sort

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
d	0	—	—	Sorts in descending order. Default is to sort in ascending order.
numeric				Sorts strings numerically

Return value

string array

Examples

```
// Demonstrate a simple sort.
string $array[3;
$array[0] = "apple";
$array[1] = "Berry;
$array[2] = "Apple";
string $sortedArray[] = `sortStrings $array`;
print $sortedArray;
```

Additional information

Related commands

- | [sortFloats \(page 1231\)](#)
- | [sortInts \(page 1233\)](#)

sqlColumnCount

Description

Summary

Retrieve the number of columns in the result set

Details

This command will return the number of columns in a result set generated by a previous call to [sqlQuery \(page 1267\)](#).

Functional area

Sql

Command syntax

Syntax

```
sqlColumnCount id
```

Arguments

Name	Type	Required	Comments
id	int	yes	Handle for the result set to a previous query

Flags

None

Return value

integer

Returns the number of columns in the query result set

Examples

```
// Get the entire result table
int $tableDump = `sqlQuery "select * from myTable"`;

// Get the number of columns in the table
int $colCount = `sqlColumnCount $tableDump`;
```

Additional information

Related commands

- [sqlColumnNames \(page 1239\)](#)
- [sqlColumnTypes \(page 1241\)](#)
- [sqlQuery \(page 1267\)](#)

sqlColumnNames

Description

Summary

Get record set column names.

Details

This command returns the names of the columns in a result set that was generated by a previous call to [sqlQuery \(page 1267\)](#).

Functional area

Sql

Command syntax

Syntax

```
sqlColumnNames id
```

Arguments

Name	Type	Required	Comments
id	int	yes	Handle for the result set to a previous query

Flags

None

Return value

string array

Array of column names from a query's result set

Examples

```
// Get the entire result table
int $tableDump = `sqlQuery "select * from myTable"`;

// Get the number of columns in the table
string $colNames[] = `sqlColumnNames $tableDump`;
```

Additional information

Related commands

- [sqlColumnCount \(page 1237\)](#)
- [sqlColumnTypes \(page 1241\)](#)
- [sqlQuery \(page 1267\)](#)

sqlColumnTypes

Description

Summary

Get the data types of a record sets columns.

Details

This command returns the types of each column in a result set that was generated by a previous call to [sqlQuery \(page 1267\)](#).

Functional area

Sql

Command syntax

Syntax

```
sqlColumnTypes id
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query

Flags

None

Return value

string array

Array of column data types, as strings, from a query's result set

Examples

```
// Get the entire result table
int $tableDump = `sqlQuery "select * from myTable"`;

// Print the columns in the table
string $colTypes[] = `sqlColumnTypes $tableDump`;
print $colTypes;
```

Additional information

Related commands

- | [sqlColumnCount \(page 1237\)](#)
- | [sqlColumnNames \(page 1239\)](#)
- | [sqlQuery \(page 1267\)](#)

sqlConnect

Description

Summary

Connect to an SQL database

Details

Use this command to establish a connection to an SQL database using either a connection string or a set of parameters (DSN, username, and password). You must supply either one connection string, or supply three strings; one for the DSN, User Name, and Password. It is safe to repeatedly call this if you are already connected so there is no need to check if you are connected, or disconnect, before attempting to connect. All subsequent calls to the SQL commands will operate on this database connection.

If you want to operate on another database, you can call sqlConnect with the connection string to connect to the other database, which will first close the current database connection for you.

Functional area

Sql

Command syntax

Syntax

```
sqlConnect ["connectionString" | "dsn" ["uid" "pwd"] ]
```

Arguments

Name	Type	Required	Comments
connectionString	string	no	Connection string sent to ODBC system
dsn	string	no	DSN (Data Source Name) to which to connect

Name	Type	Required	Comments
pwd	string	no	Password for SQL account
uid	string	no	User name for SQL account

Flags

None

Return value

void

Examples

```
// Connect to the database myDB as username John, with password 12345.
sqlConnect "myDB" "john" "12345";
// Do some stuff
// Disconnect from the database
sqlDisconnect;
```

Additional information

Related commands

■ [sqlDisconnect \(page 1245\)](#)

sqlDisconnect

Description

Summary

Disconnect from a connected SQL database

Details

After work with a connected SQL database is complete, this call will disconnect the database to free resources on the database server.

Functional area

Sql

Command syntax

Syntax

```
sqlDisconnect
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Connect to the database myDB as username John, with password 12345.  
sqlConnect "myDB" "john" "12345";  
// Do some stuff  
// Disconnect from the database  
sqlDisconnect;
```

Additional information

Related commands

■ [sqlConnect \(page 1243\)](#)

sqlExecute

Description

Summary

Execute a SQL statement

Details

This script command allows the execution of a SQL statement on a database, such as a INSERT statement. If you want to perform a query on a database, then use the [sqlQuery \(page 1267\)](#) command.

Functional area

Sql

Command syntax

Syntax

```
sqlExecute "sqlString"
```

Arguments

Name	Type	Required	Comments
sqlString	string	yes	SQL command to execute

Flags

None

Return value

integer

Returns the number of affected rows by the execute operation

Examples

```
// Connect to the database myDB as username John, with password 12345.
sqlConnect "myDB" "John" "12345";

// Execute a SQL statement, here filling in the column actor for rows
// where status is `active`.
sqlExecute "update table1 set actor=`Jeff` where status=`active`";
```

Additional information

Related commands

■ [sqlQuery \(page 1267\)](#)

sqlFirst

Description

Summary

Sets the SQL cursor to the first row in a result set

Details

Sets the SQL cursor to the first row from a result set generated by a previous SQL query.

This command acts as a sort of "rewind". Calling it is only necessary if you have already changed the cursor using [sqlNext \(page 1263\)](#) or [sqlLast \(page 1261\)](#).

By default, the cursor is set to the first row.

Functional area

Sql

Command syntax

Syntax

```
sqlFirst id
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query

Flags

None

Return value

boolean

Returns true if the cursor was successfully set at the first row, false otherwise

Examples

```
// Get the entire table
int $queryHandle = `sqlQuery "select * from myTable"`;

// Set the cursor to the first row. This actually isn't necessary as
// calling sqlQuery automatically sets the cursor to the first row for
// you.
sqlFirst $queryHandle;

// Keep iterating through all of the rows
while( `sqlNext $queryHandle` == true )
{
    // Do something with this row of data, e.g. call sqlGetString
}
```

Additional information

Related commands

- | [sqlGetRow \(page 1255\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlNext \(page 1263\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlGetFloat

Description

Summary

Retrieves the floating point value stored in a field of the selected row in a record set.

Details

Retrieves the floating point value of a field at the current SQL cursor given a specified column index or column name.

Functional area

Sql

Command syntax

Syntax

```
sqlGetFloat id ["fieldName" | fieldIndex]
```

Arguments

Name	Type	Required	Comments
id	int	yes	Handle for the result set to a previous query
fieldName	string	no	Name of the column from which to retrieve the field. Either the fieldIndex or fieldName argument must be used.
fieldIndex	int	no	Zero-based index of the column number from which to retrieve the field. Either the fieldIndex or fieldName argument must be used.

Flags

None

Return value

float

The float value stored in a field of the selected row in a record set.

Examples

```
// Get the entire result table
int $queryHandle = `sqlQuery "select * from myTable"`;

// Set the cursor to the last row
sqlLast $queryHandle;

// Get the first field for the last row
float $firstFieldLastRow = `sqlGetFloat $queryHandle 0`;
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlGetInt \(page 1253\)](#)
- | [sqlGetRow \(page 1255\)](#)
- | [sqlGetString \(page 1257\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlNext \(page 1263\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlGetInt

Description

Summary

Retrieve a single field from a record set as an integer

Details

Retrieves the integer value of a field at the current SQL cursor given a specified column index or column name.

Functional area

Sql

Command syntax

Syntax

```
sqlGetInt id ["fieldName" | fieldIndex]
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query
fieldName	string	no	Name of the column from which to retrieve the field. Either the <code>fieldIndex</code> or <code>fieldName</code> argument must be used.
fieldIndex	integer	no	Zero-based index of the column number from which to retrieve the field. Either the <code>fieldIndex</code> or <code>fieldName</code> argument must be used.

Flags

None

Return value

integer

Examples

```
// Get the entire result table
int $queryHandle = `sqlQuery "select * from myTable"`;

// Set the cursor to the last row
sqlLast $queryHandle;

// Get the first field for the last row
int $firstFieldLastRow = `sqlGetInt $queryHandle 0`;
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlGetFloat \(page 1251\)](#)
- | [sqlGetRow \(page 1255\)](#)
- | [sqlGetString \(page 1257\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlNext \(page 1263\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlGetRow

Description

Summary

Get record set row as string array.

Details

Retrieves the fields in a row at the current SQL cursor, with each string holding a value for a single column field.

Functional area

Sql

Command syntax

Syntax

```
sqlGetRow id
```

Arguments

Name	Type	Required	Comments
id	int	yes	Handle for the result set to a previous query

Flags

None

Return value

string array

The array of fields representing a row of data in a data set

Examples

```
// Get the entire result table
int $tableDump = `sqlQuery "select * from myTable"`;

// Set the cursor to the last row
sqlLast $tableDump;

// Get the fields for the last row
string $lastRow[] = `sqlGetRow $tableDump`;
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlGetFloat \(page 1251\)](#)
- | [sqlGetInt \(page 1253\)](#)
- | [sqlGetString \(page 1257\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlNext \(page 1263\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlGetString

Description

Summary

Retrieve a single field from a record set as a string

Details

Retrieves the string value of a field at the current SQL cursor given a specified column index or column name.

Functional area

Sql

Command syntax

Syntax

```
sqlGetString id ["fieldName" | fieldIndex]
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query
fieldName	string	no	Name of the column from which to retrieve the field. Either the <code>fieldIndex</code> or <code>fieldName</code> argument must be used.
fieldIndex	integer	no	Zero-based index of the column number from which to retrieve the field. Either the <code>fieldIndex</code> or <code>fieldName</code> argument must be used.

Flags

None

Return value

string

Examples

```
// Get the entire result table
int $queryHandle = `sqlQuery "select * from myTable"`;

// Set the cursor to the last row
sqlLast $queryHandle;

// Get the first field for the last row
string $firstFieldLastRow = `sqlGetString $queryHandle 0`;
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlGetFloat \(page 1251\)](#)
- | [sqlGetInt \(page 1253\)](#)
- | [sqlGetRow \(page 1255\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlNext \(page 1263\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlIsFieldNull

Description

Summary

Determines if a field in a row of a SQL result set is null.

Details

Determines if a field in a row of a SQL result set is null. Null fields have no value (e.g. a null field is not equivalent to the value of 0; it is defined as having no value).

Functional area

Sql

Command syntax

Syntax

```
sqlIsFieldNull id ["fieldName" | fieldIndex]
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query
fieldName	string	no	Name of the column from which to retrieve the field. Either the <code>fieldIndex</code> or <code>fieldName</code> must be used.
fieldIndex	integer	no	Zero-based index of the column number from which to retrieve the field. Either the <code>fieldIndex</code> or <code>fieldName</code> argument must be used.

Flags

None

Return value

boolean

Returns true if the field is empty, false if a value is set.

Examples

```
// Get the entire result table
int $queryHandle= `sqlQuery "select * from myTable"`;

// Set the cursor to the last row
sqlLast $queryHandle;

// See if the first field for the last row is empty
boolean $isNull = `sqlIsFieldNull $queryHandle 0`;
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlNext \(page 1263\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlLast

Description

Summary

Sets the SQL cursor to the last row in a result set

Details

Sets the SQL cursor to the last row in a result set generated by a previous SQL query.

Functional area

Sql

Command syntax

Syntax

```
sqlLast id
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query

Flags

None

Return value

boolean

Returns true if the cursor was successfully set at the last row, false otherwise

Examples

```
// Get the entire result table
int $queryHandle = `sqlQuery "select * from myTable"`;

// Set the cursor to the last row to operate on it.
sqlLast $queryHandle;
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlGetRow \(page 1255\)](#)
- | [sqlNext \(page 1263\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlNext

Description

Summary

Sets the SQL cursor to the next row in a result set

Details

Sets the SQL cursor to the next row in a result set generated by a previous SQL query. When this command returns false, you know you have iterated through all of the rows in your query.

Functional area

Sql

Command syntax

Syntax

```
sqlNext id
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query

Flags

None

Return value

boolean

Returns true if the cursor was successfully set to the next row, false otherwise.

Examples

```
// Get the entire table
int $queryHandle = `sqlQuery "select * from myTable"`;

// Keep iterating through all of the rows
while( `sqlNext $queryHandle` == true )
{
    // Do something with this row of data, e.g. call sqlGetString
}
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlGetRow \(page 1255\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlPrev \(page 1265\)](#)

sqlPrev

Description

Summary

Sets the SQL cursor to the previous row in a result set

Details

Sets the SQL cursor to the previous row in a result set generated by a previous SQL query.

Functional area

Sql

Command syntax

Syntax

```
sqlPrev id
```

Arguments

Name	Type	Required	Comments
id	integer	yes	Handle for the result set to a previous query

Flags

None

Return value

boolean

Returns true if the cursor was successfully set to the previous row, false otherwise

Examples

```
// Get the entire result table
int $queryHandle = `sqlQuery "select * from myTable"`;

// Iterate through all the rows in reverse
sqlLast $queryHandle;
while( `sqlPrev $queryHandle` )
{
    // Do something with this row
}
}
```

Additional information

Related commands

- | [sqlFirst \(page 1249\)](#)
- | [sqlGetRow \(page 1255\)](#)
- | [sqlLast \(page 1261\)](#)
- | [sqlNext \(page 1263\)](#)

sqlQuery

Description

Summary

Execute a SQL query

Details

Use this command to retrieve data from your database. The command produces a "result set" which you iterate through (using the [sqlNext \(page 1263\)](#) command) and access using the sqlGet*** commands.

The query string should not modify your database. If you want to modify your database, use [sqlExecute \(page 1247\)](#) instead.

Functional area

Sql

Command syntax

Syntax

```
sqlQuery "queryString"
```

Arguments

Name	Type	Required	Comments
queryString	string	yes	SQL query to execute

Flags

None

Return value

integer

Returns a "handle" to the query. Use this handle as the first argument to the other SQL commands that require it.

Examples

```
// Connect to the database myDB as username John, with password 12345.
sqlConnect "myDB" "john" "12345";
// Perform the query, which will generate the result set we will iterate
// through.

int $queryHandle = `sqlQuery "select * from myTable"`;
// Keep calling sqlNext until it return false, which indicates we have
// completed iterating through the result set.
while( `sqlNext $queryHandle` == true )
{
    // Get the values of all of the fields in the current row.
    string $row[] = `sqlGetRow $queryHandle`;
    // Do something with row
}
```

Additional information

Related commands

■ [sqlConnect \(page 1243\)](#)

squareRoot

Description

Summary

Returns the square root of a number.

Details

Returns the square root of a number. If the number is negative, the command will fail.

Functional area

Math

Command syntax

Syntax

```
squareRoot number
```

Arguments

Name	Type	Required	Comments
number	float	yes	Either a float or an int whose square root is to be returned. Cannot be negative.

Flags

None

Return value

float

Returns the square root of a positive number

Examples

```
// Compute some square roots
float $root, $value = 1000;
$root = `squareRoot 9`;
print $root;
```

```
// Should be 3.000000
$root = `squareRoot $value`;
print $root;
// Should be 31.622776
```

Additional information

Related commands

- [abs \(page 101\)](#)
- [fabs \(page 372\)](#)

step

Description

Summary

Step through animation playback by specified number of frames.

Details

Step is used for customizing playback speed in Shogun Post. Setting the number of frames argument to a greater number will result in faster playback. No arguments will result in stepping by 1 frame.

Functional area

Playback control

Command syntax

Syntax

```
step frames [-b]
```

Arguments

Name	Type	Required	Comments
frames	integer	yes	step size in frames

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
b	0	—	—	Step in the reverse direction

Return value

void

Examples

```
step 4; // Play every 4 frames in the 3D views.
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [stepKey \(page 1273\)](#)
- | [stop \(page 1275\)](#)

stepKey

Description

Summary

Goes to the next frame that has a key on the selected modules.

Details

Goes to the next frame that has a key on the selected modules. (Works like [findGap \(page 404\)](#), but doesn't wrap). Useful for data analysis scripts.

Functional area

Playback control

Command syntax

Syntax

```
stepKey [-b] [-any] [-s] [-primary]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
b	0	—	—	Steps backwards in time
any	0	—	—	Steps to the next or previous frame where X Y and Z all have keys

Name	Flag arguments	Argument type	Exclusive to	Comments
s	0	—	—	Set selected modules
primary	0	—	—	Use the primary selected modules

Return value

boolean

Examples

```
//open a sample file (*.c3d)
//
//select markers
selectByType Marker;
stepKey;
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stop \(page 1275\)](#)

stop

Description

Summary

Stop play back.

Details

Stop the playback of the animation in the scene.

Functional area

Playback control

Command syntax

Syntax

```
stop
```

Arguments

None

Flags

None

Return value

void

Examples

```
stop;
```

Additional information

Related commands

- | [animRange \(page 137\)](#)
- | [fastForward \(page 374\)](#)
- | [getAnimEnd \(page 449\)](#)
- | [getPlayEnd \(page 641\)](#)
- | [getPlayStart \(page 643\)](#)
- | [getTime \(page 701\)](#)
- | [play \(page 832\)](#)
- | [playOptions \(page 834\)](#)
- | [playRange \(page 836\)](#)
- | [replay \(page 929\)](#)
- | [setRangesFollowActiveClip \(page 1141\)](#)
- | [setTime \(page 1173\)](#)
- | [step \(page 1271\)](#)
- | [stepKey \(page 1273\)](#)

strCompare

Description

Summary

Compares two strings alphabetically.

Details

Compares two strings alphabetically. If the first string is "before" the second string, a value less than 0 is returned. If the second string is "before" the first string, a value greater than 0 is returned. If the two strings are equal, 0 is returned.

By default, the string comparison is case-sensitive (e.g. ``A`` is less than ``a``). It compares ASCII character values, with capital letters (A-Z) being before lower case letters (a-z). This can be overridden with the `-noCase` flag.

This command is useful for sorting lists of strings. Note that if you just wish to determine if two strings are identical, you may use the equality operator ``==`` on the two strings.

Functional area

String

Command syntax

Syntax

```
strCompare "string1" "string2" [-noCase]
```

Arguments

Name	Type	Required	Comments
<code>string1</code>	string	yes	First string to be compared.
<code>string2</code>	string	yes	Second string to be compared.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
noCase	0	—	—	Specifies that the search be case insensitive. The default is case sensitive.

Return value

integer

See *Details* section above for explanation of the return value.

Examples

```
// Demonstrate string comparison.
string $str1 = "Bob";
string $str2 = "Stan";
string $str3 = "bob";
int $val;

// Compare the first two strings.
$val = `strCompare $str1 $str2`;
print $val;
// Should be less than 0

// Compare the second and third strings.
$val = `strCompare $str2 $str3`;
print $val;
// Should be less than 0

// Compare the first and the third.
$val = `strCompare $str1 $str3`;print $val;
// Should be less than 0

// Do a case insensitive search of the first and third
$val = `strCompare $str1 $str3 -noCase`;
print $val;
// Should be 0
```

Additional information

Related commands

- [strFind \(page 1280\)](#)
- [strLeft \(page 1282\)](#)
- [strLength \(page 1284\)](#)
- [strMid \(page 1286\)](#)
- [strReplace \(page 1288\)](#)
- [strReverseFind \(page 1291\)](#)
- [strRight \(page 1293\)](#)
- [strTok \(page 1295\)](#)
- [strTokArray \(page 1298\)](#)

strFind

Description

Summary

Finds a sub-string within another string.

Details

Tries to find a sub-string within another string, and returns the zero-based index of the first character of the sub-string in string. If the sub-string is not found, -1 is returned.

Functional area

String

Command syntax

Syntax

```
strFind "string" "subString" index
```

Arguments

Name	Type	Required	Comments
string	string	yes	String to be searched.
subString	string	yes	Sub-string to be searched for with "string" argument.
startIndex	integer	yes	Zero-based index of the character of string to start searching at, or 0 to search from the beginning.

Flags

None

Return value

integer

Returns the zero based index of the location of the first character of the sub-string found.
Returns -1 if not found.

Examples

```
// Search for a substring
string $str = "This is some text";
int $index;

// Search for the word "some"
$index = `strFind $str "some" 0`;
print $index;
// Should be 8

// Search for the word "texas"
$index = `strFind $str "texas" 0`;
print $index;
// Should be -1

// Search for the letter "t" after the word "some"
$index = `strFind $str "t" 12`;
print $index;
// Should be 13
```

Additional information

Related commands

- [strCompare \(page 1277\)](#)
- [strLeft \(page 1282\)](#)
- [strLength \(page 1284\)](#)
- [strMid \(page 1286\)](#)
- [strReplace \(page 1288\)](#)
- [strReverseFind \(page 1291\)](#)
- [strRight \(page 1293\)](#)
- [strTok \(page 1295\)](#)
- [strTokArray \(page 1298\)](#)

strLeft

Description

Summary

Extracts a sub-string from the beginning of another string.

Details

Extracts the first (left-most) "count" characters from "string". If "count" is greater than the length of string, the entire string is returned.

Functional area

String

Command syntax

Syntax

```
strLeft "string" count
```

Arguments

Name	Type	Required	Comments
string	string	yes	String to extract sub-string from.
count	integer	yes	The number of characters to extract from string, counting from the left.

Flags

None

Return value

string

Returns the sub-string from "string".

Examples

```
// Extract some substrings
string $str = "This is some text";
string $subStr;

// Extract the first 7 characters
$subStr = `strLeft $str 7`;
print $subStr;
// Should be "This is"

// Extract the first 17 characters
$subStr = `strLeft $str 17`;
print $subStr;
// Should be same as $str
```

Additional information

Related commands

- | [strCompare \(page 1277\)](#)
- | [strFind \(page 1280\)](#)
- | [strLength \(page 1284\)](#)
- | [strMid \(page 1286\)](#)
- | [strReplace \(page 1288\)](#)
- | [strReverseFind \(page 1291\)](#)
- | [strRight \(page 1293\)](#)
- | [strTok \(page 1295\)](#)
- | [strTokArray \(page 1298\)](#)

strLength

Description

Summary

Returns the length of a string.

Details

Returns the length of string. The length of a string is the number of ASCII characters (not to be confused with the character module in Shogun Post) contained within the string.

Functional area

String

Command syntax

Syntax

```
strLength "string"
```

Arguments

Name	Type	Required	Comments
string	string	yes	String to get the length of.

Flags

None

Return value

integer

Returns the number of characters in the string.

Examples

```
// Get the length of a string
string $str = "This is some text";
int $length;

// Length of str
$length = `strLength $str`;
print $length;
// Should be 17

// Length of an empty string
$length = `strLength ""`;
print $length;
// Should be 0
```

Additional information

Related commands

- | [strCompare \(page 1277\)](#)
- | [strFind \(page 1280\)](#)
- | [strLeft \(page 1282\)](#)
- | [strMid \(page 1286\)](#)
- | [strReplace \(page 1288\)](#)
- | [strReverseFind \(page 1291\)](#)
- | [strRight \(page 1293\)](#)
- | [strTok \(page 1295\)](#)
- | [strTokArray \(page 1298\)](#)

strMid

Description

Summary

Extracts a sub-string from the middle of another string.

Details

Extracts a sub-string of `count` characters from `string`, starting at position `firstIndex`. If `count` is -1, the remainder of string is extracted.

Functional area

String

Command syntax

Syntax

```
strMid "string" firstIndex count
```

Arguments

Name	Type	Required	Comments
<code>string</code>	string	yes	String to extract sub-string from.
<code>firstIndex</code>	integer	yes	The zero-based index of the first character in string that is to be extracted as a sub-string.
<code>count</code>	integer	yes	The number of characters to extract from string, counting from <code>firstIndex</code> .

Flags

None

Return value

string

Returns the sub-string from a string.

Examples

```
// Demonstrate usage of strMid
string $str = "This is some text";
string $subStr;

// Get 5 characters starting at index 3
$subStr = `strMid $str 3 5`;
print $subStr;
// Should be "s is "

// Get rest of string from index 8
$subStr = `strMid $str 8 -1`;
print $subStr;
// Should be "some text"
```

Additional information

Related commands

- | [strCompare \(page 1277\)](#)
- | [strFind \(page 1280\)](#)
- | [strLeft \(page 1282\)](#)
- | [strLength \(page 1284\)](#)
- | [strReplace \(page 1288\)](#)
- | [strReverseFind \(page 1291\)](#)
- | [strRight \(page 1293\)](#)
- | [strTok \(page 1295\)](#)
- | [strTokArray \(page 1298\)](#)

strReplace

Description

Summary

Replaces a sub-string with another sub-string in a string.

Details

Replaces all instances of a sub-string within a string with another string, and returns the resulting string.

The search for `subStringOld` is case-sensitive. `subStringOld` and `subStringNew` do not have to be the same length.

To remove all occurrences of `subStringOld`, make `subStringNew` an empty string (`""`).

Functional area

String

Command syntax

Syntax

```
strReplace "string" "subStringOld" "subStringNew"
```

Arguments

Name	Type	Required	Comments
<code>string</code>	string	yes	String to replace sub-strings in.
<code>subStringOld</code>	string	yes	Sub-string to be replaced in string. Can be 1 or more characters in length.
<code>subStringNew</code>	string	yes	String to replace all instances of <code>subStringOld</code> in string. Can be 0 or more characters in length.

Flags

None

Return value

string

Returns a new string with the old values replaced with new.

Examples

```
// Demonstrate usage of strReplace
string $str = "This is some text";
string $newStr;

// Replace "some" with "a bit of"
$newStr = `strReplace $str "some" "a bit of"`;
print $newStr;
// Should be "This is a bit of text"

// Remove all occurrences of "i" in the new string
$newStr = `strReplace $newStr "i" ""`;
print $newStr;
// Should be "Ths s a bt of text"

// Replace all x's with s's
$newStr = `strReplace $str "x" "s"`;
print $newStr;
// Should be "This is some test"
```

Additional information

Related commands

- [strCompare \(page 1277\)](#)
- [strFind \(page 1280\)](#)
- [strLeft \(page 1282\)](#)
- [strLength \(page 1284\)](#)
- [strMid \(page 1286\)](#)
- [strReverseFind \(page 1291\)](#)
- [strRight \(page 1293\)](#)
- [strTok \(page 1295\)](#)
- [strTokArray \(page 1298\)](#)

strReverseFind

Description

Summary

Searches a string in reverse order for a character.

Details

Searches a string in reverse order for a character.

Because Shogun Post does not have a character data type, a string is used. If character string is empty, the command will fail. If the character string is longer than one character, only the first character will be used in the reverse search.

Functional area

String

Command syntax

Syntax

```
strReverseFind "string" "findChar"
```

Arguments

Name	Type	Required	Comments
string	string	yes	String to search in.
char	string	yes	String of length 1, which will be searched for in string.

Flags

None

Return value

integer

Returns the zero-based index of the last occurrence a character in a string.

Examples

```
// Search for a character in reverse order. Extract
// the string after the character.
string $str = "This is some text";
string $subStr;
int $index, $length;

// Search in reverse for a space
$index = `strReverseFind $str " "`;
print $index;
// Should be 12

// Get the remainder of the string after that.
$length = `strLength $str`;
$index = $length - $index - 1;
$subStr = `strRight $str $index`;
print $subStr;
// Should be "text"
```

Additional information

Related commands

- | [strCompare \(page 1277\)](#)
- | [strFind \(page 1280\)](#)
- | [strLeft \(page 1282\)](#)
- | [strLength \(page 1284\)](#)
- | [strMid \(page 1286\)](#)
- | [strReplace \(page 1288\)](#)
- | [strRight \(page 1293\)](#)
- | [strTok \(page 1295\)](#)
- | [strTokArray \(page 1298\)](#)

strRight

Description

Summary

Extracts a sub-string from the end of another string.

Details

Extracts the last (right-most) `count` characters from "`string`". If `count` is greater than the length of "`string`", the entire string is returned.

Functional area

String

Command syntax

Syntax

```
strRight "string" count
```

Arguments

Name	Type	Required	Comments
<code>string</code>	string	yes	String to extract sub-string from.
<code>count</code>	integer	yes	The number of characters to extract from string, counting from the right.

Flags

None

Return value

string

Returns the sub-string from "string"

Examples

```
// Extract some substrings
string $str = "This is some text";
string $subStr;

// Extract the last 9 characters
$subStr = `strRight $str 9`;
print $subStr;
// Should be "some text"

// Extract the last 17 characters
$subStr = `strRight $str 17`;
print $subStr;
// Should be same as $str
```

Additional information

Related commands

- | [strCompare \(page 1277\)](#)
- | [strFind \(page 1280\)](#)
- | [strLeft \(page 1282\)](#)
- | [strLength \(page 1284\)](#)
- | [strMid \(page 1286\)](#)
- | [strReplace \(page 1288\)](#)
- | [strReverseFind \(page 1291\)](#)
- | [strTok \(page 1295\)](#)
- | [strTokArray \(page 1298\)](#)

strTok

Description

Summary

Returns an element of a string as a "token".

Details

Tokenizing a string is the process of breaking apart a string into components that are meaningful to you. For example, if whole words are the components we are concerned with in a sentence, then we would expect white-space characters to separate, or delimit, the words of the sentence. Thus, in this scenario, the sentence "Here is some text" would be tokenized into "Here", "is", "some", and "text".

To tokenize a string, first call `strTok` with the `strToTokenize` argument, which "seeds" the tokenizer for subsequent calls to `strTok`. It also returns the first token, if one is found.

Then call `strTok` repeatedly to get the elements of the string, until an empty string is returned.

The default delimiters are the white-space characters (i.e. space, tab, and new line characters), but any delimiters may be used.

To match the full delimiter string (which will almost never be done with the default delimiters), specify the `-matchAll` flag.

This command is similar to the [strTokArray \(page 1298\)](#) command, except that `strTokArray` tokenizes the string in one call, returning an array of all the elements. Use `strTokArray` if you won't change your delimiter in the middle of tokenizing.

Functional area

String

Command syntax

Syntax

```
strTok "string"[-delims string] [-matchAll]
```

Arguments

Name	Type	Required	Comments
strToTokenize	string	no	The string to be tokenized. Only needs to be provided on the first call to strTok. Subsequent calls will operate on this string, until this argument is provided again.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
delims	1	string	—	Overrides the default delimiters with user provided delimiters. The default delimiters are space, tab, and new line (any white space).
matchAll	0	—	—	Specifies that all the delimiter characters must be matched. The default is to match any of the delimiter characters.

Return value

string

Returns the "token" found with "string", or an empty string if the end of the string has been reached.

Examples

```
// Demonstrate string tokenization.
string $str = "This is some text";
string $filepath = "c:/Program Files/ShogunPost#.#/SampleFiles/Day1/Test.hdf";
string $token;

// Tokenize the first string. Start off by
// "seeding" the strTok command with str. It
// will return the first token if one was found
$token = `strTok $str`;

// Keep tokenizing until we have no more tokens
```



```
while( `strLength $token` != 0 )
{
    // Print out our token
    print $token;
    // Get the next token. don't pass in $str, as
    // we only need to seed it in the beginning
    $token = `strTok`;
}
// Now tokenize the file path. Tokenize it by
// folders, so use / as a delimiter
$token = `strTok $filepath -delims "/"`;
while( `strLength $token` != 0 )
{
    // Print out our token
    print $token;
    // Get the next token. don't pass in $filepath, as
    // we only need to seed it in the beginning
    $token = `strTok -delims "/"`;
}
```

Additional information

Related commands

- | [strCompare \(page 1277\)](#)
- | [strFind \(page 1280\)](#)
- | [strLeft \(page 1282\)](#)
- | [strLength \(page 1284\)](#)
- | [strMid \(page 1286\)](#)
- | [strReplace \(page 1288\)](#)
- | [strReverseFind \(page 1291\)](#)
- | [strRight \(page 1293\)](#)
- | [strTokArray \(page 1298\)](#)

strTokArray

Description

Summary

Returns elements of a string as "tokens".

Details

Tokenizing a string is the process of breaking apart a string into components that are meaningful to you. For example, if whole words are the components we are concerned with in a sentence, then we would expect white-space characters to separate, or delimit, the words of the sentence. Thus, in this scenario, the sentence "Here is some text" would be tokenized into "Here", "is", "some", and "text". To tokenize a string, call `strTokArray` with the string to be tokenized.

The default delimiters are the white-space characters (i.e. space, tab, and new line characters), but any delimiters may be used.

To match the full delimiter string (which will almost never be done with the default delimiters), specify the `-matchAll` flag.

This command is similar to the [strTok \(page 1295\)](#) command, except that `strTok` requires multiple calls to tokenize the string, returning one token of the string with each call. Use `strTok` if you need to change your delimiter in the middle of tokenizing.

Functional area

String

Command syntax

Syntax

```
strTokArray "string" [-delims string] [-matchAll]
```

Arguments

Name	Type	Required	Comments
strToTokenize	string	yes	The string to be tokenized.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
delims	1	string	—	Overrides the default delimiters with user provided delimiters. The default delimiters are space, tab, and new line (any white space).
matchAll	0	—	—	Specifies that all the delimiter characters must be matched. The default is to match any of the delimiter characters.

Return value

string array

Returns all of the "tokens" found within "string".

Examples

```
// Demonstrate string tokenization.
string $str = "This is some text";
string $filepath = "C:/Program Files/ShogunPost#.#/SampleFiles/Day1/Test.hdf";
string $tokens[];

// Tokenize the first string.
$tokens = `strTokArray $str`;
print $tokens;

// Now tokenize the file path. Tokenize it by
// folders, so use / as a delimiter
$tokens = `strTokArray $filepath -delims "/"`;
print $tokens;
```

Additional information

Related commands

- | [strCompare \(page 1277\)](#)
- | [strFind \(page 1280\)](#)
- | [strLeft \(page 1282\)](#)
- | [strLength \(page 1284\)](#)
- | [strMid \(page 1286\)](#)
- | [strReplace \(page 1288\)](#)
- | [strReverseFind \(page 1291\)](#)
- | [strRight \(page 1293\)](#)
- | [strTok \(page 1295\)](#)

swap

Description

Summary

Swap marker translation data on two indicated markers beginning at the current frame through the end of the file.

Details

When editing marker data, often you will come across "swaps". These are a by-product of optical motion capture systems when the definition (label) of a marker suddenly (over a single frame, or following a gap) swaps its label with a neighboring marker. This is generally brought about by a physical proximity of two markers in excess of the system's ability to resolve.

Shogun's swap command provides options to swiftly remedy these occurrences whether they occur over the entire frame range of the file, or over a subset of the frame range.

Functional area

Data manipulators

Command syntax

Syntax

```
swap [-all] [-ranges] [-diff] [-currentFrame]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	ranges	Swaps data over entire file
ranges	0	—	all	Swaps data over the selected ranges only
diff				
currentFrame				

Return value

void

Examples

```
select LFWT LWRI;
swap -ranges;
// Assuming that you have identified the time frame(s) where swaps
// occurred between these two markers, the swap command will fix them
swap;swap -all;
// Performs a swap from the current frame to the start of the move
```

Additional information

Related commands

- [copyPattern \(page 226\)](#)
- [findBadData \(page 395\)](#)

system

Description

Summary

Allows you to start processes or run DOS commands.

Details

The system command allows you to start processes/applications, or allows you to execute DOS commands when using the `-dos` flag.

One way to think of the distinction is to consider the difference between entering values in the Run dialog box, compared with a DOS command prompt.

Functional area

System

Command syntax

Syntax

```
system "argList" ["startDir"] [-async] [-hide] [-captureOutput] [-dos] [-noFeedback] [-title string]
```

Arguments

Name	Type	Required	Comments
argList	string	yes	When using the <code>-dos</code> flag, this argument is the DOS command you wish to execute. Otherwise this is a string composed of the name of the application you wish you launch (using a full path if the application does not exist in one of the folders in your PATH environment variable) as well as any values you wish to pass to the application as startup/command line parameters. See <i>Examples</i> section below.

Name	Type	Required	Comments
startDir	string	no	Allows you to specify the starting "working" directory for the process. By default, the starting directory is the directory which the process is located in. This argument isn't used when you use the -dos flag.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
async	0	—	captureOutput	Set this flag to launch the process in a separate thread so that Shogun Post will not get locked up while the program is running. By default, Shogun Post waits for the process (or DOS command) to finish.
hide	0	—	dos	Set this flag to launch the process with the process' window hidden.
captureOutput	0	—	async, dos	For processes that are console applications, this flag will redirect the output normally sent to the console window to Shogun Post's command log. This output will also be used as the return value for the command.
dos	0	—	hide, captureOutput	This flag causes the system command to invoke the DOS command interpreter through the C Runtime system() command. In other words, use this flag if you wish to execute one of the available commands of the DOS command interpreter (e. g. the dir del commands. By

Name	Flag arguments	Argument type	Exclusive to	Comments
				default, the system command will try and execute a process through the CreateProcess Windows API.
noFeedback	0	—	async	If the captureOutput flag is set, set this flag to avoid writing the output to the Shogun Post log file.
title	1	string	hide, dos	If the type of process being launched is a console application, this flag will set the title of the console window. This flag has no effect on GUI processes.

Return value

string

Returns the captured output of the command if the -captureOutput flag is specified.

Examples

```
// Show how to use the -dos option, which will delete a file on disk.
// Even though Shogun Post deals with file paths using forward slashes,
// the DOS command interpreter expects back-slashes. This means that you
// will need to use double-back slashes in the string you pass to the
// command since the back-slash is an "escape character" in Shogun Post,
// meaning it is used in combination with the next character to signify
// a special value. In addition, if the file path has a space in it,
// you will need to surround it with quotes as in this example here
// (the same goes if we didn't specify the -dos flag)
system "del \"C:\\PathToSome\\File.dat\" \" \" -dos;

// Execute a console application and redirect its output to the Shogun Post
// command prompt.
system "\"C:\\My\\ConsoleApp.exe\" SomeValue AnotherValue" -captureOutput";

// This launches Notepad. Using the -async flag allows the script
// that calls the system command to continue executing. Without it,
// Shogun Post would be hung until the user closed the Notepad window.
system "notepad.exe C:\\SomeFile.txt" -async;
```

tabWindow

Description

Summary

Add window to tab

Details

Allows you to group docking windows into tabbed groups. If the other docking window is not already part of a tabbed group, a tabbed group is created and both docking windows will be part of the new tabbed group.

Functional area

Interface

Command syntax

Syntax

```
tabWindow "windowName" "WindowToTabWithName"
```

Arguments

Name	Type	Required	Comments
windowName	string	yes	The name of the docking window to create a tab for
WindowToTabWithName	string	yes	The name of the other docking window to which the window will be tabbed.

Flags

None

Return value

void

Examples

```
// This command adds the "Attributes" window to the tabbed group
// that the "Channels" window is part of. If the Channels window is not
// part of a tabbed group, one will be created and both windows will be
// added to it
tabWindow "Attributes" "Channels";
```

Additional information

Related commands

- | [autohideWindow \(page 156\)](#)
- | [dockWindow \(page 361\)](#)
- | [floatWindow \(page 428\)](#)
- | [setWindowSize \(page 1191\)](#)
- | [showWindow \(page 1197\)](#)

tan

Description

Summary

Returns the tangent of an input angle (degrees)

Details

The tan command is used to calculate the tangent for any input angle. The input units are in degrees.

Picture the unit circle (a circle of radius 1.0 centered about the origin). If you were to imagine a unit vector rotating inside of the unit circle, the angle (in degrees) that the unit vector makes with the positive x-axis is the input argument theta. The slope of the unit vector is the return value of the tan function.

Functional area

Math

Command syntax

Syntax

```
tan value
```

Arguments

Name	Type	Required	Comments
value	float	yes	Float value of an angle in degrees.

Flags

None

Return value

float

Returns a floating point value. tan is undefined at {..., -270, -90, 90, 270, ...}

Examples

```
float $temp = (tan (30));  
// declares a variable for the tangent function  
  
print (string ($temp));  
// prints the output of the tangent function
```

Additional information

Related commands

- | [acos \(page 103\)](#)
- | [asin \(page 144\)](#)
- | [atan \(page 148\)](#)
- | [cos \(page 230\)](#)
- | [sin \(page 1199\)](#)

testPrint

Description

Summary

Test print command

Functional area

Testing

Command syntax

Syntax

```
testPrint
```

Arguments

None

Flags

None

Return value

void

tileClips

Description

Summary

Tiles the clips in time by adjusting their offset

Details

From top to bottom the clips are tiled end to end. Useful when setting up clips for blending.

Functional area

Data manipulators

Command syntax

Syntax

```
tileClips
```

Arguments

None

Flags

None

Return value

void

Examples

```
// Load 2 c3d files and tile their clips  
loadFile "C:/Take0001.c3d";  
loadFile "C:/Take0002.c3d";  
tileClips;
```

Additional information

Related commands

- | [addLayer \(page 108\)](#)
- | [addToClip \(page 129\)](#)
- | [copyClip \(page 219\)](#)
- | [cropClip \(page 313\)](#)
- | [deleteClipData \(page 334\)](#)
- | [flatten \(page 426\)](#)
- | [getActiveLayer \(page 437\)](#)
- | [getClips \(page 471\)](#)
- | [moveClip \(page 803\)](#)
- | [offsetClips \(page 820\)](#)
- | [removeFromClip \(page 903\)](#)
- | [removeLayer \(page 905\)](#)
- | [resetClip \(page 931\)](#)
- | [selectClipObjects \(page 979\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveClip \(page 1017\)](#)
- | [setActiveLayer \(page 1020\)](#)

tPoseLabel

Description

Summary

Labels an actor standing in a T-pose using an uncalibrated skeleton.

Details

Labels an actor or multiple actors standing in a T-pose using an uncalibrated skeleton. If no characters are selected, all are T-pose labeled. If one or more characters are selected, only they are T-pose labeled.

For best results the pose of the skeleton and actor should be the same.

Multiple actors can be processed but they should stand a reasonable distance apart and the separation distance may need to be adjusted accordingly.

Functional area

Labeling

Command syntax

Syntax

```
tPoseLabel
```

Arguments

None

Flags

None

Return value

void

Examples

```
// T pose label  
tPoseLabel;
```

Additional information

Related commands

■ [tPoseLabelOptions \(page 1315\)](#)

tPoseLabelOptions

Description

Summary

Specifies the options to be used during execution of the [tPoseLabel \(page 1313\)](#) command.

Details

Specifies the options to be used during execution of the tPoseLabel command. See *Flags* comments below for descriptions of their use.

Functional area

Labeling

Command syntax

Syntax

```
tPoseLabelOptions [-standardDeviation float] [-separationDistance float]  
[-currentFrameOnly boolean] [-autoLabel boolean] [-reset]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
standardDeviation	1	float	—	Specifies the expected error in marker placement between the VST and actors markers as a standard deviation of their distance from each other.
separationDistance	1	float	—	Specifies the expected distance between the actors when TPose label is performed when multiple actors are in the scene.
currentFrameOnly	1	boolean	—	Applies the labels to the current frame only.
reset	0	—	—	Resets the options to default values.

Return value

void

Examples

```
// Increase the standard deviation to account for marker placement
// on the actor not matching the VST very well.
tPoseLabelOptions -standardDeviation 50;
```

Additional information

Related commands

■ [tPoseLabel \(page 1313\)](#)

transcodeVideo

Description

Summary

This command performs Bayer conversion of raw video captured from Vicon Bonita and Vue video cameras if needed and optionally compresses the video.

Details

If you do not specify a codec, the output video file is uncompressed. To include a codec in your Shogun Post script, you must first install and configure the codec on a PC that is used by Shogun Post.

Functional area

Data editing

Command syntax

Syntax

```
transcodeVideo "sourceFile" "destinationFile" [-codec string] [-
timecodeOverlay] [-timecodeOverlayPosition string] [-timecodeOverlaySize
integer]
```

Arguments

Name	Type	Required	Comments
sourceFile	string	Yes	The full path and filename of the raw video file that is to be converted.
destinationFile	string	Yes	The full path and filename of the converted file.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
codec	1	string	—	The name of the codec to be used for the transcode, which must be a 64-bit codec that is installed on the PC used by Shogun Post. If transcodeVideo is called with an invalid codec, a list of available codecs is displayed in the Log.
timecodeOverlay			—	Specifies timecode overlay that occurs during transcoding
timecodeOverlay Position	1	string	—	Specifies the location for timecode overlay. Options match those in the Batching panel.
timecodeOverlay Size	1	integer	—	Specifies the font size of the timecode overlay. If this is not specified, 24 (the default value) is used.

Return value

void

Examples

```
// The following command converts a raw video file called
// Take_001.avi, saves the output .avi to a location
// different from that of the original file, and specifies the
// DV Video Encoder codec to compress it.
transcodeVideo "F:/Video/Take_001.avi" "D:/Transcoded_Data/Take_001.avi" -codec
"DV Video Encoder";
```

trcExportOptions

Description

Summary

Set TRC export options

Functional area

File handling

Command syntax

Syntax

```
trcExportOptions [-writeSubjectNames boolean] [-removeLeadingUnderscore  
boolean] [-writeUnlabeled boolean] [-collapseSubjects boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
writeSubjectNames	1	boolean	—	—
removeLeadingUnderscore	1	boolean	—	—
writeUnlabeled	1	boolean	—	—
collapseSubjects	1	boolean	—	—

Return value

Void

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dImportOptions \(page 1375\)](#)
- | [xcplImportOptions \(page 1377\)](#)

unbreakTangents

Description

Summary

Unbreak the tangent on a selected key into a single part or handle manipulator.

Details

When broken, the tangent manipulators can be used to adjust curve slopes independently on either side of the key.

An unbroken tangent does not allow the curve to be adjusted independently on either side of a key.

Functional area

Data manipulators

Command syntax

Syntax

```
unbreakTangents
```

Arguments

None

Flags

None

Return value

void

Additional information

Related commands

- [breakTangents \(page 177\)](#)
- [createTangents \(page 303\)](#)

uncalibrateLabelingCluster

Description

Summary

Uncalibrates the selected labeling cluster.

Details

Functional area

Labeling

Command syntax

Syntax

```
uncalibrateLabelingCluster
```

Arguments

None

Flags

None

Return value

void

undo

Description

Summary

Allows you to return your scene file to its state immediately prior to a single previous operation, or many recent operations.

Details

Undo is a staple command of virtually all computer software. Shogun Post maintains a history stack of 20 operations, and enables you to undo in one step an optional [nLevels] number of those operations.

Functional area

System

Command syntax

Syntax

```
undo [depth][-enable] [-disable]
```

Arguments

Name	Type	Required	Comments
depth		No	Specifies the number of previous operations to undo in one step

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
enable	0	—	disable	—
disable	0	—	enable	—

Return value

void

Examples

```
undo; //undo the previous operation
undo 6; // undo the previous 6 operations.
```

Additional information

Related commands

- [bakeData \(page 172\)](#)
- [redo \(page 891\)](#)
- [undoConsolidated \(page 1326\)](#)

undoConsolidated

Description

Summary

Turns on or off consolidated undo.

Details

If you turn on `undoConsolidated`, the state of the current scene is saved to a temporary `.vdf` file, which acts as a restore point, and the standard undo stack processing is turned off.

When `undoConsolidated` is turned off, an undo item, which references the temporary `.vdf` file, is added to the undo stack.

If you undo the `undoConsolidated` command, you are prompted to confirm that you want to replace the scene.

Functional area

System

Command syntax

Syntax

```
undoConsolidated on/off/toggle
```

Return value

void

Additional information

Related commands

- redo (page 891)
- undo (page 1324)

unlabel

Description

Summary

Remove labels from markers

Details

Removes labels from markers by moving translation keys to a new unlabeled marker.

The new unlabeled marker's parent can be specified with the `-parent` flag.

Functional area

Data manipulators

Command syntax

Syntax

```
unlabel [-parent string] [-diff] [-ranges] [-all] [-currentFrame]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
parent	1	string	—	Specifies the module to which unlabeled markers should be parented.
diff				
ranges				
all				
currentFrame				

Return value

void

Examples

```
// Unlabel LFHD and parent it to the myUnlabeledMarkers module
unlabel "LFHD" -parent "myUnlabeledMarkers";
```

Additional information

Related commands

■ [label \(page 763\)](#)

unpack

Description

Summary

Unpack marker data

Details

This command unpacks previously packed trajectory data. It operates on selected unlabeled trajectories and creates a new unlabeled trajectory for each contiguous key range encountered.

Note that any gaps in trajectories are lost when unpacking.

Functional area

Data editing

Command syntax

Syntax

```
unpack
```

Arguments

None

Flags

None

Return value

void

Examples

```
// This selects and unpacks all unlabeled trajectories  
selectByName "_" -type Marker;  
unpack;
```

Additional information

Related commands

■ [pack \(page 824\)](#)

unparent

Description

Summary

Make the selected nodes top-level nodes (removes any parents of the selected objects).

Details

The unparent command enables you to disassemble hierarchies among scene objects.

The transformation values of the selected object(s) remain unchanged after the parent operation unless the `-adjust` option is used. Therefore, if there are any non-zero transform values above the selected node(s), you must use `unparent -adjust` to maintain the node's absolute world location and orientation subsequent to execution of the command.

The parent of the selected objects does not need to be selected to unparent objects. However, the converse is not true; the target parent does need to be selected when parenting selected objects.

Functional area

Data manipulators

Command syntax

Syntax

```
unparent [-adjust] [-showProgress] [-ranges] [-adjustOnly]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
adjust	0	—	adjustOnly	Recalculates the transform of the nodes to account for any previous parent transform.
showProgress	0	—	—	—
ranges	0	—	—	—
adjustOnly	0	—	adjust	—

Return value

void

Examples

```
select AUX1 AUX2 AUX3 AUX4 Prop_rigid;
unparent -adjust;
// Unparent the prop markers from the Prop_rigid (rigid Body)
// and adjust for any transformations that may exist on Actor_1.
```

Additional information

Related commands

■ [parent \(page 826\)](#)

unsetProperty

Description

Summary

Unsets optional attributes.

Details

Some attributes have the ability to be considered unset, meaning they are not set to any value. Typically such attributes are considered optional. This command unsets such attributes.

Functional area

Data manipulators

Command syntax

Syntax

```
unsetProperty propertyName [-all] [-onMod string] [-type string]
```

Arguments

See [setProperty \(page 1130\)](#) command.

Flags

See [setProperty \(page 1130\)](#) command.

Return value

void

Additional information

Related commands

- [setProperty \(page 1130\)](#)

updateClip

Description

Summary

Update a clip/blend

Details

NLE clips, like loops and simple blends, need to be updated any time one of their source clip(s) has their animation data changed. For example, a if you change a key on a marker in the clip that a loop is looping, then the loop will need to be updated, or re-evaluated. Normally, this happens for you as you are changing values in the GUI. However, if you change animation data during a script, then the loop is not updated until the script has finished executing. This enables you to update the loop (or other blend type) while a script is executing.

This command is useful if you need to query the blend for a value after you have changed some animation data.

Functional area

Data manipulators

Command syntax

Syntax

```
updateClip ["clip"]
```

Arguments

Name	Type	Required	Comments
clip	string	no	The name of the clip to update. If not specified, then the active clip is updated.

Flags

None

Return value

void

Examples

```
// Set a random key on the selected marker. Then update the Loop
setKey "Translation.X" 100;
updateClip "MyLoop";
```

Additional information

Related commands

- | [getActiveClip \(page 435\)](#)
- | [setActiveClip \(page 1017\)](#)

velocityLabel

Description

Summary

Velocity labels the selected marker.

Details

Velocity labels the selected marker by looking for markers adjacent in time whose velocity matches the selected marker within various adjustable tolerances.

Functional area

Labeling

Command syntax

Syntax

```
velocityLabel
```

Arguments

None

Flags

None

Return value

integer

Returns the number of keys labeled.

Examples

```
// Velocity label the LFHD marker.  
select LFHD;  
velocityLabel;
```

Additional information

Related commands

■ [velocityLabelOptions \(page 1339\)](#)

velocityLabelOptions

Description

Summary

Sets the options for the [velocityLabel \(page 1337\)](#) command.

Functional area

Labeling

Command syntax

Syntax

```
velocityLabelOptions [-startupError float] [-predictionError float] [-  
maxGap integer] [-numAvgFrames integer] [-frameByFrame boolean] [-  
stopOnMultiple boolean] [-continuous boolean] [-primary boolean][-reset]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
startupError	1	float	—	Specifies the max distance in mm between the marker being labeled and other markers for it to be considered for labeling.
predictionError	1	float	—	Specifies the max velocity between the marker being labeled and other markers for it to be considered for labeling. Measured in mm per second.
maxGap	1	integer	—	Specifies the max number of frames without data the velocity labeler will continue trying to label before quitting.
numAvgFrames	1	integer	—	Specifies the number of frames to use when computing the average velocity of the marker we are labeling.
frameByFrame	1	boolean	—	Specifies if the labeler should test consider each frame or only each trajectory.
stopOnMultiple	1	boolean	—	Specifies if the labeler should stop when multiple solutions are present.
continuous	1	boolean	—	Specifies if the labeler should iterate until no more labels are made.
primary	1	boolean	—	
reset	0	—	—	Resets the options to default values.

Return value

void

Examples

```
// Set the velocity labeler to be able to skip gaps smaller than  
// 5 frames.  
velocityLabelOptions -maxGap 5;
```

Additional information

Related commands

■ [velocityLabel \(page 1337\)](#)

viewLayout

Description

Summary

Changes the current interface layout to one of eight indexed layouts or changes a current view to any of the four available view types.

Details

The viewLayout command is used for setting view types and view layouts. Using the -v options, you may configure the preset layouts to contain named layout elements in each of the views defined by any layout.

The number of views ranges from one to four. The views are numbered left to right, with the first (-v1) being located in the upper left, the fourth (-v4) located in the lower right.

The argument for the type of viewLayout is optional if any of the flags are used.

Functional area

Interface

Command syntax

Syntax

```
viewLayout viewTypeIndex [-v1 string] [-v2 string] [-v3 string] [-v4  
string] [-fullwindow boolean] [-swap] [-current string] [-active integer]
```

Argument

Name	Type	Required	Comments
viewTypeIndex			The arguments are an integer value layout number from 1 to 8 indicating the desired view layout.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
v1	1	string	—	Set viewport 1 to specified type
v2	1	string	—	Set viewport 2 to specified type
v3	1	string	—	Set viewport 3 to specified type
v4	1	string	—	Set viewport 4 to specified type
full window	1	boolean	—	Set active view to/fromfull window
swap	0	—	—	Swaps to previous view layout
current	1	string	—	Set the current view
active	1	integer	—	—
fullwindow	1	boolean	—	—

Return value

void

Examples

```
viewLayout 5;
viewLayout -v1 perspective -v2 graph -v3 vpl;
// The first line in the above command configures the view layout to
// contain 3 viewports.
// The second line establishes a Perspective view, a Graph view,
// and a VPL view.
// Note that second command does not require an argument
```

Additional information

Related commands

- [cameraView \(page 196\)](#)
- [camerasView \(page 202\)](#)
- [graphView \(page 730\)](#)

vskExportOptions

Description

Summary

Set export options for vsk files.

Details

Set export options for Vicon skeleton (vsk) files. When this type of the file is saved the options affect the output format.

Functional area

File handling

Command syntax

Syntax

```
vskExportOptions [-version string]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
version	—	string	—	—

Return value

void

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dImportOptions \(page 1375\)](#)
- | [xcplImportOptions \(page 1377\)](#)

wait

Description

Summary

Waits for a specified time.

Details

The wait command is used in a script to cause a time delay, specified in milliseconds. It could be used after a command that is known to take a while to complete, and which is needed to be complete for further commands in the script, for example, waiting for the system to connect.

Functional area

Interface

Command syntax

Syntax

```
wait milliseconds
```

Arguments

Name	Type	Required	Comments
milliseconds	int	yes	The time that the script will pause on the command, eg, 3000 is 3 seconds.

Flags

None

Return value

void

Examples

```
//print to log the time at the beginning and end of the script
int $TimeNow = `getSystemTime`;
print $TimeNow;
wait 3000;
$TimeNow = `getSystemTime`;print $TimeNow;
//The log shows 2 numbers that have a difference of 3
//meaning 3 seconds have elapsed
```

windowExists

Description

Summary

Determines whether or not a given docking window exists.

Details

This is useful for custom GUI work inside of Shogun Post. This command can be used to check if a specified user window exists. This could be used before creating a user window to check if creating or destroying a window with a specified name would cause an error.

Functional area

User Window

Command syntax

Syntax

```
windowExists "windowName"
```

Arguments

Name	Type	Required	Comments
windowName	string	yes	The name of the user window.

Flags

None

Return value

boolean

Examples

```
//code from AutoPropVST script
//check at start of operation for existence of the AutoPropVST window
//if it exists, destroy it so that it can be recreated in default form
if(`windowExists "AutoPropVST"`)
{
    destroyWindow "AutoPropVST";
}
```

Additional information

Related commands

- [createWindow \(page 311\)](#)
- [destroyWindow \(page 358\)](#)
- [getWindowVisibility \(page 725\)](#)
- [showWindow \(page 1197\)](#)

writeBool

Description

Summary

Writes one or more Boolean values to a file

Details

Use to write one or more Boolean values to a file. This command accepts any number of Boolean arguments, including arrays.

The following list shows the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
"c"	Writes Boolean as a signed character, or a signed 8 bit (1 byte) value
"uc"	Writes Boolean as an unsigned character, or a unsigned 8 bit (1 byte) value
"s"	Writes Boolean as a signed short integer, or a signed 16 bit (2 bytes) value
"us"	Writes Boolean as an unsigned short integer, or an unsigned 16 bit (2 bytes) value
"i"	Writes Boolean as a signed integer, or a signed 32 bit (4 bytes) value
"ui"	Writes Boolean as an unsigned integer, or an unsigned 32 bit (4 bytes) value
"f"	Writes Boolean as a floating point value, a 32 bit (4 byte) value

Functional area

Disk I/O

Command syntax

Syntax

```
writeBool fileID boolVal1 [boolVal2...] [-cast string]
```

Arguments

Name	Type	Required	Comments
boolean2			Optional Boolean or Boolean array: Command can accept any number of Booleans/arrays to write to the file
boolean1			Boolean or Boolean array: 1 or more values to be written to the file
fileID	int		ID of file previously opened with fileOpen (page 387) .

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast to. Possible types are listed below. Note that data can be lost when casting data.

Return value

void

Examples

```
//Write example
boolean $val = true;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "w"`;

// Write the boolean value
writeBool $fileID $val;
fileClose $fileID;
```

Additional information

Related commands

- | [writeFloat \(page 1354\)](#)
- | [writeInt \(page 1357\)](#)
- | [writeRot \(page 1366\)](#)
- | [writeString \(page 1369\)](#)
- | [writeVec \(page 1372\)](#)

writeFloat

Description

Summary

Writes one or more floating point values to a file

Details

Use to write one or more floating point values to a file. This command will accept any number of floating point arguments, including arrays.

The following list shows the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
"c"	Writes float as a signed character, or a signed 8 bit (1 byte) value
"uc"	Writes float as an unsigned character, or an unsigned 8 bit (1 byte) value
"s"	Writes float as a signed short integer, or a signed 16 bit (2 bytes) value
"us"	Writes float as an unsigned short integer, or an unsigned 16 bit (2 bytes) value
"i"	Writes float as a signed integer, or a signed 32 bit (4 bytes) value
"ui"	Writes float as an unsigned integer, or an unsigned 32 bit (4 bytes) value
"f"	Writes float as a floating point value, a 32 bit (4 byte) value

Functional area

Disk I/O

Command syntax

Syntax

```
writeFloat fileID floatVal1 [floatVal2...] [-cast string]
```

Arguments

Name	Type	Required	Comments
float2	optional float or float array		Command can accept any number of floats /arrays to write to the file
float1	float or float array		One or more values to be written to the file
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast to. Possible types are listed above. Note that data can be lost when casting data.

Return value

void

Examples

```
//Write example
float $val = 5.75;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "w"`;

// Write the float value
writeFloat $fileID $val;
fileClose $fileID;
```

Additional information

Related commands

- | [writeBool \(page 1351\)](#)
- | [writeInt \(page 1357\)](#)
- | [writeRot \(page 1366\)](#)
- | [writeString \(page 1369\)](#)
- | [writeVec \(page 1372\)](#)

writeln

Description

Summary

Writes one or more integer values to a file

Details

Use to write one or more integer values to a file. This command will accept any number of integer arguments, including arrays.

The following list shows the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
"c"	Writes integer as a signed character, or a signed 8 bit (1 byte) value
"uc"	Writes integer as an unsigned character, or an unsigned 8 bit (1 byte) value
"s"	Writes integer as a signed short integer, or a signed 16 bit (2 bytes) value
"us"	Writes integer as an unsigned short integer, or an unsigned 16 bit (2 bytes) value
"i"	Writes integer as a signed integer, or a signed 32 bit (4 bytes) value
"ui"	Writes integer as an unsigned integer, or an unsigned 32 bit (4 bytes) value
"f"	Writes integer as a floating point value, a 32 bit (4 byte) value

Functional area

Disk I/O

Command syntax

Syntax

```
writeInt fileID intVal1 [intVal2...] [-cast string]
```

Arguments

Name	Type	Required	Comments
int2	optional int or int array		Command can accept any number of ints/arrays to write to the file
int1	int or int array		One or more values to be written to the file
fileID	int		ID of file previously opened with fileOpen

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast to. Possible types are listed below. Note that data can be lost when casting data.

Return value

void

Examples

```
//Write example
int $val = 50;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "w"`;

// Write the int value
writeInt $fileID $val;
fileClose $fileID;
```

Additional information

Related commands

- | [writeBool \(page 1351\)](#)
- | [writeFloat \(page 1354\)](#)
- | [writeRot \(page 1366\)](#)
- | [writeString \(page 1369\)](#)
- | [writeVec \(page 1372\)](#)

writeProfileInt

Description

Summary

Writes an integer value to the user's profile.

Details

Profile values are identified by their section (sections begin with [section name]) and by the entry (or key) name.

Section groupings allow values to be organized in logical groups. They also allow values with the same name to be differentiated between different groups.

Specifying `-file` allows the values to be written to a file of the user's choice.

The file does not have to have an "ini" extension.

Some users use the profile (or other file) as a "global variable" mechanism. While this is still supported, it is not recommended and has been superseded by actual global variables which allow you to have variables that persist between script executions. For more information, see the [setGlobalVar \(page 1074\)](#) command (and related commands).

Functional area

System

Command syntax

Syntax

```
writeProfileInt "sectionHeader" "entry" value [-file string]
```


Arguments

Name	Type	Required	Comments
sectionHeader	string	yes	Logical grouping of the value. The section header is the part of the profile which is enclosed in square brackets (e.g. [My Section])
entry	string	yes	The entry name. This will be the first value on the line followed by an = sign.
value	integer	yes	Value to write to the profile.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Redirects output to another text file. The default is to output to the user's profile (.ini file).

Return value

boolean

Returns true if the value was successfully written, otherwise false.

Examples

```
// Store some information in the profile
int $value;

// Write to Section 1
writeProfileInt "Section1" "Value1" 10;
writeProfileInt "Section1" "Value2" -20;

// Write to Section 2
writeProfileInt "Section2" "Value1" 10;
writeProfileInt "Section2" "Value2" 20;

/* Will produce output like this
[Section1]
```

```
Value1=10
Value2=-20
[Section2]
Value1=10
Value2=20
*/

// Now retrieve it
$value = `getProfileInt "Section1" "Value1" 0`;
print $value;
```

Additional information

Related commands

- | [getProfileInt \(page 651\)](#)
- | [getProfileString \(page 653\)](#)
- | [writeProfileString \(page 1363\)](#)

writeProfileString

Description

Summary

Writes a string value to the users profile.

Details

Profile values are identified by their section (sections begin with [section name]) and by the entry (or key) name.

Section groupings allow values to be organized in logical groups. They also allow values with the same name to be differentiated between different groups.

Specifying `-file` allows the values to be written to a file of the user's choice.

The file does not have to have an "ini" extension.

Note that some users use the profile (or other file) as a "global variable" mechanism. While this is still supported, it is not recommended and has been superseded by actual global variables which allow you to have variables that persist between script executions. For more information, see [setGlobalVar \(page 1074\)](#) (and related commands).

Functional area

System

Command syntax

Syntax

```
writeProfileString "sectionHeader" "entry" "value" [-file string]
```

Arguments

Name	Type	Required	Comments
sectionHeader	string	yes	Logical grouping of the value. The section header is the part of the profile which is enclosed in square brackets (e.g. [My Section])
entry	string	yes	The entry name. This will be the first value on the line followed by an = sign.
value	string	yes	Value to write to the profile.

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
file	1	string	—	Redirects output to another text file. The default is to output to the user's profile (.ini file).

Return value

boolean

Returns true if the value was successfully written, otherwise false.

Examples

```
// Store some information in the profile
string $value;
// Write to Section 1
writeProfileString "Section1" "Value1" "Hello";
writeProfileString "Section1" "Value2" "World";
// Write to Section 2
writeProfileString "Section2" "Value1" "Scripting";
writeProfileString "Section2" "Value2" "is cool";

/* Will produce output like this
[Section1]
Value1=Hello
Value2=World
```

```
[Section2]
Value1=Scripting
Value2=is cool
*/

// Now retrieve it
$value = `getProfileString "Section1" "Value1" "`;
print $value;
```

Additional information

Related commands

- | [getProfileInt \(page 651\)](#)
- | [getProfileString \(page 653\)](#)
- | [writeProfileInt \(page 1360\)](#)

writeRot

Description

Summary

Writes one or more rotations to a file

Details

Use to write one or more rotation values to a file. This command accepts any number of vector arguments, including arrays.

The following list shows the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
"c"	Writes each rotation channel value as a signed character, or a signed 8 bit (1 byte) value
"uc"	Writes each rotation channel value as an unsigned character, or an unsigned 8 bit (1 byte) value
"s"	Writes each rotation channel value as a signed short integer, or a signed 16 bit (2 bytes) value
"us"	Writes each rotation channel value as an unsigned short integer, or an unsigned 16 bit (2 bytes) value
"i"	Writes each rotation channel value as a signed integer, or a signed 32 bit (4 bytes) value
"ui"	Writes each rotation channel value as an unsigned integer, or an unsigned 32 bit (4 bytes) value
"f"	Writes each rotation channel value as a floating point value, a 32 bit (4 byte) value

Functional area

Disk I/O

Command syntax

Syntax

```
writeRot fileID rotValue1 [rotValue2...] [-delim string] [-quat] [-rotOrder string] [-cast string] [-rad]
```

Arguments

Name	Type	Required	Comments
rotation2	optional vector or vector array		Command can accept any number of vectors /arrays to write to the file
rotation1			Vector or vector array: 1 or more values to be written to the file
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
delim	1	string	—	Delimiter to use between rotation channel values. Cannot be used with binary files. Usually ",", or "\t", etc.
quat	0	—	rotOrder	Specifies that the rotation should be written as a quaternion (four float values) rather than an Euler angle (three float values). Can not be used with -rotOrder option
rotOrder	1	string	quat	Specifies the rotation order that the rotation will be written in. Can not be used with -quat option (possible values are XYZ, XZY, YXZ, YZX, ZXY, and ZYX)

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast to. Possible types are listed below. Note that data can be lost when casting data.
rad	0	—	quat	—

Return value

void

Examples

```
//Rot write example
vector $rot = << 25.0, -70.5, 19 >>;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "w"`;

// Write the rotation value as an Euler angle
// The -delims flag assumes you are writing to a text file
writeRot $fileID $rot -delim ",";
fileClose $fileID;
```

Additional information

Related commands

- [writeBool \(page 1351\)](#)
- [writeFloat \(page 1354\)](#)
- [writeInt \(page 1357\)](#)
- [writeString \(page 1369\)](#)
- [writeVec \(page 1372\)](#)

writeString

Description

Summary

Writes one or more strings to a file

Details

Use to write one or more strings to a file. Command will accept any number of string arguments, including arrays.

The following is a listing of the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Casting doesn't make sense when writing strings, but the option is still there for you.

Cast	Description
"c"	Writes each string character as a signed character, or a signed 8 bit (1 byte) value
"uc"	Writes each string character as an unsigned character, or an unsigned 8 bit (1 byte) value
"s"	Writes each string character as a signed short integer, or a signed 16 bit (2 bytes) value
"us"	Writes each string character as an unsigned short integer, or an unsigned 16 bit (2 bytes) value
"i"	Writes each string character as a signed integer, or a signed 32 bit (4 bytes) value
"ui"	Writes each string character as an unsigned integer, or an unsigned 32 bit (4 bytes) value
"f"	Writes each string character as a floating point value, a 32 bit (4 byte) value

Functional area

Disk I/O

Command syntax

Syntax

```
writeString fileID stringVal1 [stringVal2...] [-cast string] [-length integer]
```

Arguments

Name	Type	Required	Comments
string 2	String or string array	No	Command can accept any number of strings/arrays to write to the file
string 1	String or string array	No	1 or more string to be written to the file
fileID	int		ID of file previously opened with fileOpen (page 387)

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
cast	1	string	—	Data type to cast to. Possible types are listed above. Note that data can be lost when casting data.
length	1	integer	—	Length of the string to write. Strings are usually written out at a fixed length, or prepended with the length of the string if the string length is variable. If you want to write out a fixed length string, use this flag. Max length is 1024

Return value

void

Examples

```
//Write example
string $str = "This is some text";
int $length;
int $fileID;
int $fileID = `fileOpen "C:/FileTesting.txt" "w"`;

// Get the string length
$length = `strLength $str`;

// First write out the string length
writeInt $fileID $length;

// Then write the string
writeString $fileID $str;
fileClose $fileID;
```

Additional information

Related commands

- [writeBool \(page 1351\)](#)
- [writeFloat \(page 1354\)](#)
- [writeRot \(page 1366\)](#)
- [writeRot \(page 1366\)](#)
- [writeVec \(page 1372\)](#)

writeVec

Description

Summary

Writes one or more vectors to a file

Details

Use to write one or more vectors to a file. Command will accept any number of vector arguments, including arrays.

The following list shows the different cast types. Note that data can be lost when casting. 'Signed' means the value can be negative. 'Unsigned' means value will always be positive.

Cast	Description
"c"	Writes each vector channel value as a signed character, or a signed 8 bit (1 byte) value"
"uc"	Writes each vector channel value as an unsigned character, or an unsigned 8 bit (1 byte) value
"s"	Writes each vector channel value as a signed short integer, or a signed 16 bit (2 bytes) value
"us"	Writes each vector channel value as an unsigned short integer, or an unsigned 16 bit (2 bytes) value
"i"	Writes each vector channel value as a signed integer, or a signed 32 bit (4 bytes) value
"ui"	Writes each vector channel value as an unsigned integer, or an unsigned 32 bit (4 bytes) value
"f"	Writes each vector channel value as a floating point value, a 32 bit (4 byte) value

Functional area

Disk I/O

Command syntax

Syntax

```
writeVec "fileID" vecValue1 vecValue2...[-delim string] [-cast string]
```

Arguments

Name	Type	Required	Comments
fileID	int		ID of file previously opened with fileOpen (page 387)
vector1	vector or vector array		One or more values to be written to the file
vector2	optional vector or vector array		Command can accept any number of vectors /arrays to write to the file

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
delim	1	string	—	Delimiter to use between vector channel values. Cannot be used with binary files. Usually "," or "\t", etc.
cast	1	string	—	Data type to cast to. Possible types are listed above. Note that data can be lost when casting data.

Return value

void

Examples

```
vector $vec = << 100.0, 500.7, -50.5 >>;
int $fileID;
// Open the file and get the ID
// ...
// Write the vector
// The -delims flag assumes you are writing to a text file
writeVec $fileID $vec -delims ",";
```

Additional information

Related commands

- | [writeBool \(page 1351\)](#)
- | [writeFloat \(page 1354\)](#)
- | [writeInt \(page 1357\)](#)
- | [writeRot \(page 1366\)](#)
- | [writeString \(page 1369\)](#)

x2dImportOptions

Description

Summary

Set X2D import options

Details

Set X2D import options. When X2D file is imported the options affect the imported objects.

Functional area

File handling

Command syntax

Syntax

```
x2dImportOptions [-importCal boolean] [-importSubjects boolean] [-importHardwareSettings boolean] [-importVideo boolean] [-movevideo boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
importCal	1	boolean	—	Import/Skip associated camera calibration file from ENF file.

Name	Flag arguments	Argument type	Exclusive to	Comments
<code>importSubjects</code>	1	boolean	—	Import/Skip associated characters from ENF file.
<code>importHardwareSettings</code>		boolean		
<code>importVideo</code>		boolean		
<code>movevideo</code>		boolean		

Return value

void

Examples

```
// Import X2D file with the cameras. Do not import subjects.
x2dImportOptions -importCal on -importSubjects off;
loadFile -createSecondFig -importType "selCreateNew" "C:/ShogunPost/Thursday
July20/00001/0000134.x2d";
```

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [mcpImportOptions \(page 798\)](#)
- | [saveFile \(page 943\)](#)
- | [xcplImportOptions \(page 1377\)](#)

xcplImportOptions

Description

Summary

Set XCP import options

Details

Set import options for extended camera parameter file. When this type of the file is imported the options affect the imported objects.

Functional area

File handling

Command syntax

Syntax

```
xcplImportOptions [-importCal boolean] [-importThreshold boolean]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
importCPFile	1	boolean	—	Import/Skip camera parameters.
importVTFile	1	boolean	—	Import/Skip threshold parameters.

Name	Flag arguments	Argument type	Exclusive to	Comments
importCal	1	boolean	—	—
importThreshold	1	boolean	—	—

Return value

void

Examples

```
// Import extended camera parameter file with the cameras.
// Do not import threshold map.
xcpImportOptions -importCPFile on -importVTTFFile off;
loadFile -createSecondFig -importType "selCreateNew" "C:/ShogunPost/Thursday
July20/00001/00001.xcp";
```

Additional information

Related commands

- | [amcExportOptions \(page 135\)](#)
- | [bvhExportOptions \(page 179\)](#)
- | [c3dExportOptions \(page 181\)](#)
- | [cpExportOptions \(page 232\)](#)
- | [fbxExportOptions \(page 376\)](#)
- | [fbxImportOptions \(page 380\)](#)
- | [loadFile \(page 779\)](#)
- | [mcpExportOptions \(page 796\)](#)
- | [mcpImportOptions \(page 798\)](#)
- | [saveFile \(page 943\)](#)
- | [x2dImportOptions \(page 1375\)](#)

zeroKey

Description

Summary

Set key to match base value.

Details

This command sets a key on the selected module to the base value (default is zero). The operation can be applied to the selected ranges or to the entire play range using the appropriate flags, or just the current frame if no flags are supplied.

Functional area

Data editing

Command syntax

Syntax

```
zeroKey [-primaryOnly] [-ranges] [-all]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
primaryOnly	0	—	—	Operate on the primary selected module only
ranges	0	—	all	Set the keys to the base value only on the frames within the selected ranges
all	0	—	ranges	Set the keys to the base value on the whole play range

Return value

void

Examples

```
// Select a marker
select Marker_1;
zeroKeys -all;
```

zoomView

Description

Summary

Zoom active view so that all objects are visible.

Details

This command resets the navigational state of the viewport to frame all of the objects in the scene. Optionally, it frames just the current selection.

Functional area

Interface

Command syntax

Syntax

```
zoomView [-all] [-selected]
```

Arguments

None

Flags

Name	Flag arguments	Argument type	Exclusive to	Comments
all	0	—	—	Zoom all views
selected	0	—	—	Zoom on selected objects in the view only

Return value

void

Examples

```
zoomView -all;  
// Frame all objects in all visible views.
```

Additional information

Related commands

■ [viewLayout \(page 1342\)](#)

Commands by category

From the following lists, you can access Vicon Shogun Post script commands by category.

- | [Data editing commands \(page 1384\)](#)
- | [Data management \(Eclipse\) commands \(page 1385\)](#)
- | [Data manipulators commands \(page 1385\)](#)
- | [Data retrieval commands \(page 1386\)](#)
- | [Data streaming commands \(page 1388\)](#)
- | [Disk I/O commands \(page 1389\)](#)
- | [File handling commands \(page 1390\)](#)
- | [Interface commands \(page 1390\)](#)
- | [Labeling commands \(page 1392\)](#)
- | [Master/Slave \(DBS\) commands \(page 1392\)](#)
- | [Math commands \(page 1393\)](#)
- | [Namespace commands \(page 1393\)](#)
- | [NLE \(Non-Linear Editor\) commands \(page 1394\)](#)
- | [Parameters commands \(page 1394\)](#)
- | [Playback control commands \(page 1395\)](#)
- | [Remote control commands \(page 1395\)](#)
- | [Selection commands \(page 1396\)](#)
- | [Skeletal solving commands \(page 1397\)](#)
- | [Snap commands \(page 1398\)](#)
- | [SQL commands \(page 1398\)](#)
- | [String commands \(page 1399\)](#)
- | [System commands \(page 1399\)](#)
- | [Testing commands \(page 1401\)](#)
- | [User window commands \(page 1401\)](#)

Data editing commands

Use the Vicon Shogun Post script commands in this category to edit your mocap data.

Data editing commands

axiomLabel (page 167)	label (page 763)
axiomLabelOptions (page 169)	labelOptions (page 765)
bakeData (page 172)	makeUnique (page 788)
bakeLengths (page 175)	pack (page 824)
collapse (page 214)	pasteKeys (page 828)
copyData (page 222)	quickPost (page 849)
copyKeys (page 224)	quickPostLabelOptions (page 851)
copyPattern (page 226)	quickPostOcclusionFixOptions (page 853)
create (page 234)	quickPostSolveOptions (page 854)
createBoneVirts (page 237)	reassemble (page 879)
createKey (page 256)	reconstruct (page 882)
createReconstructorScript (page 277)	reconstructChunk (page 884)
cutKeys (page 318)	reconstructOptions (page 887)
delete (page 322)	reduceKeys (page 893)
deleteCharacters (page 332)	removeMarkerConnection (page 909)
deleteRedundant (page 344)	restoreGaps (page 933)
exists (page 367)	rtDropFix (page 940)
fillGaps (page 389)	setMarkerConnection (page 1106)
continued...	

Data editing commands

filter (page 392)	setTimeCodeOffset (page 1180)
fixOcclusion (page 422)	snapToSystemAlign (page 1223)
fixOcclusionOptions (page 424)	transcodeVideo (page 1317)
getTimeCodeOffset (page 706)	unpack (page 1329)
grayscalefitOptions (page 733)	zeroKey (page 1379)

Data management (Eclipse) commands

Use the Vicon Shogun Post script commands in this category to control data management.

Data management commands

getEclipseActiveTrial (page 515)	getSession (page 519)
getEclipseAssociatedCalibration (page 517)	openEclipseDatabase (page 822)
getEclipseAssociatedSubjects (page 519)	refreshEclipse (page 895)
getEclipseMarkedTrials (page 521)	setSession (page 1157)

Data manipulators commands

Use the Vicon Shogun script commands in this category to work with data.

Data manipulators commands

addToClip (page 129)	selectTails (page 1009)
alignAxis (page 132)	setActiveClip (page 1017)
breakTangents (page 177)	setBoneLength (page 1029)
copyClip (page 219)	setInterpType (page 1081)

continued...

Data manipulators commands

createTangents (page 303)	setKey (page 1083)
cropClip (page 313)	setPosition (page 1123)
deleteClipData (page 334)	setPriority (page 1128)
deleteTangents (page 354)	setProperty (page 1130)
getPriority (page 649)	setRotation (page 1143)
hide (page 741)	setScale (page 1147)
makeRigid (page 785)	show (page 1193)
moveClip (page 803)	swap (page 1301)
offsetClips (page 820)	tileClips (page 1311)
parent (page 826)	unbreakTangents (page 1321)
removeFromClip (page 903)	unlabel (page 1327)
removeRayContributions (page 915)	unparent (page 1331)
removeSoftwareFitCentroids (page 917)	unsetProperty (page 1333)
resetClip (page 931)	updateClip (page 1335)
rigidBody (page 937)	

Data retrieval commands

Use the Vicon Shogun Post script commands in this category to retrieve data.

Data retrieval commands

delGlobalVar (page 356)	getIntArrayProperty (page 562)
findBadData (page 395)	getIntProperty (page 564)

continued...

Data retrieval commands

findDeviantKeys (page 398)	getKeys (page 566)
findGap (page 404)	getLabelerChars (page 568)
findNonRigid (page 410)	getLabelingClusterOffsets (page 570)
findPlaneCrossing (page 412)	getLastFile (page 571)
findSelectedKeys (page 414)	getLogFile (page 592)
findTail (page 418)	getMarkerConnection (page 594)
findUnlabeled (page 420)	getMarkerConnectionColor (page 596)
getActiveClip (page 435)	getModule (page 598)
getAnimEnd (page 449)	getModuleRange (page 600)
getAnimStart (page 451)	getModules (page 602)
getAttached (page 453)	getModuleType (page 604)
getAttachedTo (page 455)	getNumKeys (page 614)
getBooleanArrayProperty (page 459)	getNumModules (page 623)
getBooleanProperty (page 461)	getParent (page 639)
getChannels (page 463)	getPlayEnd (page 641)
getChildren (page 467)	getPlayStart (page 643)
getClips (page 471)	getPosition (page 647)
getColor (page 473)	getProperty (page 655)
getConstraintError (page 477)	getRate (page 660)
getConstraintOffset (page 479)	getRotation (page 662)
continued...	

Data retrieval commands

getConstraintPos (page 481)	getScale (page 666)
getConstraintsThisIsSource (page 483)	getSelectedKeys (page 672)
getConstraintsThisIsTarget (page 485)	getSelectedTimeRanges (page 674)
getContributingCameras (page 487)	getSolverBones (page 687)
getCount (page 497)	getStringArrayProperty (page 691)
getCurrentChar (page 499)	getStringProperty (page 693)
getCurrentLabel (page 501)	getTime (page 701)
getDistance (page 507)	getTimecode (page 705)
getFloatProperty (page 543)	getTrajectories (page 713)
getGaps (page 547)	getVectorProperty (page 717)
getGlobalBooleanVar (page 550)	hasKey (page 734)
getGlobalFloatVar (page 552)	isSelected (page 755)
getGlobalIntVar (page 554)	offlineCameraHealthCheck (page 816)
getGlobalStringVar (page 556)	setGlobalVar (page 1074)
getGlobalVectorVar (page 560)	

Data streaming commands

Use the Vicon Shogun Post script commands in this category to control data streaming.

Data streaming commands

offlineDataProvider (page 819)
--

Disk I/O commands

Use the Vicon Shogun Post script commands in this category for data input and output

Disk I/O commands

fileClose (page 385)	readRot (page 866)
fileOpen (page 387)	readString (page 869)
getFilePath (page 535)	readToken (page 872)
getFilePos (page 537)	readVec (page 874)
getFiles (page 539)	readWord (page 877)
isEndOfFile (page 747)	seekToString (page 954)
isFileBinary (page 749)	setFilePos (page 1068)
isFileReadable (page 751)	writeBool (page 1351)
isFileWriteable (page 753)	writeFloat (page 1354)
readBool (page 855)	writeInt (page 1357)
readFloat (page 858)	writeRot (page 1366)
readInt (page 861)	writeString (page 1369)
readLine (page 864)	writeVec (page 1372)

File handling commands

Use the Vicon Shogun Post script commands in this category for file handling.

File handling commands

amcExportOptions (page 135)	mcpExportOptions (page 796)
asfExportOptions (page 142)	mcplImportOptions (page 798)
bvhExportOptions (page 179)	newFile (page 812)
c3dExportOptions (page 181)	saveFile (page 943)
cpExportOptions (page 232)	trcExportOptions (page 1319)
fbxExportOptions (page 376)	vskExportOptions (page 1345)
fbxImportOptions (page 380)	x2dlImportOptions (page 1375)
loadFile (page 779)	xcplImportOptions (page 1377)

Interface commands

Use the Vicon Shogun Post script commands in this category to work with the Shogun user interface.

User interface commands

applInfo (page 140)	graphView (page 730)
autohideWindow (page 156)	imageLabeler (page 743)
cameraView (page 196)	isAxiomEnabled (page 746)
camerasView (page 202)	loadScript (page 782)
closeScript (page 212)	loadWorkspaceString (page 783)
createShelfButton (page 282)	manipulator (page 789)

continued...

User interface commands

createShelfGroup (page 285)	messagePrompt (page 800)
createShelfSeparator (page 287)	progressBar (page 844)
createShelfTab (page 289)	renameShelfGroup (page 923)
dataHealthView (page 320)	renameShelfTab (page 925)
dataIssuesMap (page 321)	saveScript (page 945)
deleteShelfGroup (page 348)	saveWorkspaceFile (page 947)
deleteShelfTab (page 350)	sceneView (page 950)
dockWindow (page 361)	selectShelfTab (page 1005)
floatWindow (page 428)	setButtonShelfFile (page 1031)
getActiveShelfTab (page 439)	setEditTool (page 1061)
getActiveView (page 443)	setPinned (page 1122)
getActiveViewType (page 445)	setSelectionFilter (page 1153)
getCurrentScript (page 503)	setSolversFirst (page 1158)
getEditTool (page 523)	setTrackList (page 1183)
getNumShelfTabs (page 625)	setViewFilter (page 1187)
getSelectionFilter (page 676)	setWindowSize (page 1191)
getShelfTabs (page 685)	tabWindow (page 1306)
getViewFilter (page 719)	viewLayout (page 1342)
getViewLayout (page 720)	wait (page 1347)
getViewTypes (page 721)	zoomView (page 1381)

Labeling commands

Use the Vicon Shogun Post script commands in this category to work with labeling.

Labeling commands

autoCreateLabelingClusters (page 152)	createPropVST (page 269)
autoLabel (page 158)	createPropVSTOptions (page 270)
autoLabelOptions (page 161)	kinematicLabelOptions (page 759)
autoVST (page 165)	kinLabel (page 761)
autoVSTOptions (page 166)	removeMarker (page 907)
calibrateCharacter (page 187)	scaleCharacter (page 949)
calibrateCharacterOptions (page 189)	tPoseLabel (page 1313)
calibrateLabelingCluster (page 192)	tPoseLabelOptions (page 1315)
calibrateLabelingClusterOptions (page 193)	uncalibrateLabelingCluster (page 1323)
calibrateProp (page 194)	velocityLabel (page 1337)
calibratePropOptions (page 195)	velocityLabelOptions (page 1339)
createLabelingCluster (page 258)	

Master/Slave (DBS) commands

Use the Vicon Shogun Post script commands in this category to enable or disable the DBS (Distributed Batch System) master and/or slave.

- | [master \(page 794\)](#)
- | [slave \(page 1203\)](#)

Math commands

Use the Vicon Shogun Post script commands in this category to work with math-related information.

Math commands

■ abs (page 101)	■ getAngleTo (page 447)
■ acos (page 103)	■ getLength (page 576)
■ asin (page 144)	■ getPointClosestTo (page 645)
■ atan (page 148)	■ normalize (page 814)
■ calcIntersection (page 185)	■ setLength (page 1086)
■ cos (page 230)	■ sin (page 1199)
■ cross (page 316)	■ squareRoot (page 1269)
■ dot (page 363)	■ tan (page 1308)
■ fabs (page 372)	

Namespace commands

Use the Vicon Shogun Post script commands in this category to work with namespaces.

Namespace commands

■ addNamespace (page 117)	■ removeNamespace (page 911)
■ getNamespace (page 606)	■ replaceNamespace (page 927)
■ getNamespaces (page 608)	

NLE (Non-Linear Editor) commands

Use the Vicon Shogun Post script commands in this category to work with layers and clips.

NLE commands

- | | |
|---|--|
| ■ addLayer (page 108) | ■ removeLayer (page 905) |
| ■ flatten (page 426) | ■ selectClipObjects (page 979) |
| ■ getActiveLayer (page 437) | ■ setActiveLayer (page 1020) |
| ■ getLayers (page 574) | |

Parameters commands

Use the Vicon Shogun Post script commands in this category to work with parameters.

Parameters commands

- | | |
|---|--|
| ■ addParameter (page 119) | ■ removeAllParameters (page 899) |
| ■ addStaticParameter (page 125) | ■ removeParameter (page 913) |
| ■ getParameter (page 627) | ■ removeUnusedParameters (page 919) |
| ■ getParameterExpression (page 629) | ■ renameParameter (page 921) |
| ■ getParameterPriorValue (page 631) | ■ selectByParameter (page 968) |
| ■ getParameters (page 633) | ■ setParameter (page 1118) |
| ■ getParameterType (page 635) | ■ setParameterPrior (page 1120) |
| ■ getParameterUserValue (page 637) | ■ setStaticParameterExpression (page 1163) |
| ■ hasParameter (page 737) | ■ setStaticParameterUserValue (page 1165) |
| ■ listParameters (page 777) | |

Playback control commands

Use the Vicon Shogun Post script commands in this category to control playback of mocap data.

Playback control commands

- | | |
|--|---|
| animRange (page 137) | rewind (page 935) |
| fastForward (page 374) | setRangesFollowActiveClip (page 1141) |
| play (page 832) | setTime (page 1173) |
| playOptions (page 834) | step (page 1271) |
| playRange (page 836) | stepKey (page 1273) |
| replay (page 929) | stop (page 1275) |
-

Remote control commands

Use the Vicon Shogun Post script commands in this category to control communications between Shogun and a remote server.

Remote control commands

- | | |
|--|--|
| client (page 210) | sendRemote (page 1013) |
| remoteControl (page 897) | server (page 1015) |
-

Selection commands

Use the Vicon Shogun Post script commands in this category to work with selection.

Selection commands

getSelectionSetNodes (page 678)	selectByType (page 975)
getSelectionSets (page 680)	selectChildren (page 977)
getSelectionSetSets (page 682)	selectionSet (page 984)
isSelectionSet (page 757)	selectKeys (page 987)
select (page 956)	selectParent (page 994)
selectabilityOn (page 959)	selectProperty (page 996)
selectBranch (page 961)	selectRange (page 998)
selectByMarkerRadius (page 963)	selectRelatedKeys (page 1001)
selectByName (page 965)	selectSet (page 1003)
selectByPart (page 970)	selectTree (page 1011)
selectBySide (page 973)	setPrimary (page 1126)

Skeletal solving commands

Use the Vicon Shogun Post script commands in this category for solving marker data to a skeletal format.

Skeletal solving commands

attach (page 150)	moveRotKeyToPreRotCmd (page 806)
autoCreateSolver (page 154)	moveTransKeyToPreTransCmd (page 807)
createSkelScript (page 291)	multiplyJointRange (page 808)
createSolvingSetup (page 293)	multiplyJointStiffness (page 810)
createSolvingSetupFromLabelingSetup (page 294)	removeBone (page 901)
createSolvingSetupOptions (page 295)	setConstraint (page 1043)
extrapolateFingers (page 371)	setExtrapolateFingersAfterSolve (page 1067)
getExtrapolateFingersAfterSolve (page 525)	setPreferredPose (page 1125)
getSolversFirst (page 689)	setUseAxiomSolving (page 1185)
getUseAxiomSolving (page 715)	skelSetup (page 1201)
goToBasePose (page 727)	solve (page 1226)
goToPreferredPose (page 728)	solver (page 1228)
insertBone (page 745)	

Snap commands

Use the Vicon Shogun Post script commands in this category to snap markers or nodes to specified locations.

Snap commands

- | | |
|--|--|
| snapTo (page 1207) | snapToLocal (page 1214) |
| snapToConstraint (page 1209) | snapToRigid (page 1217) |
| snapToLine (page 1212) | snapToSystem (page 1220) |

SQL commands

Use the Vicon Shogun Post script commands in this category to manage data in a SQL database connected to Shogun.

SQL commands

- | | |
|--|--|
| sqlColumnCount (page 1237) | sqlGetInt (page 1253) |
| sqlColumnNames (page 1239) | sqlGetRow (page 1255) |
| sqlColumnTypes (page 1241) | sqlGetString (page 1257) |
| sqlConnect (page 1243) | sqlIsFieldNull (page 1259) |
| sqlDisconnect (page 1245) | sqlLast (page 1261) |
| sqlExecute (page 1247) | sqlNext (page 1263) |
| sqlFirst (page 1249) | sqlPrev (page 1265) |
| sqlGetFloat (page 1251) | sqlQuery (page 1267) |

String commands

Use the Vicon Shogun Post script commands in this category to handle strings in mocap data.

String commands

| [strCompare \(page 1277\)](#) | [strReplace \(page 1288\)](#)

| [strFind \(page 1280\)](#) | [strReverseFind \(page 1291\)](#)

| [strLeft \(page 1282\)](#) | [strRight \(page 1293\)](#)

| [strLength \(page 1284\)](#) | [strTok \(page 1295\)](#)

| [strMid \(page 1286\)](#) | [strTokArray \(page 1298\)](#)

System commands

Use the Vicon Shogun Post script commands in this category to interact with the Vicon system and Shogun.

System commands

| [addScript \(page 121\)](#) | [pathExists \(page 830\)](#)

| [addScriptPath \(page 123\)](#) | [playSound \(page 839\)](#)

| [assert \(page 146\)](#) | [print \(page 841\)](#)

| [clearLog \(page 206\)](#) | [processFile \(page 843\)](#)

| [clearScriptPaths \(page 208\)](#) | [python \(page 848\)](#)

| [copyString \(page 228\)](#) | [redo \(page 891\)](#)

| [createDir \(page 245\)](#) | [runScript \(page 941\)](#)

| [createSceneScript \(page 279\)](#) | [scriptExists \(page 952\)](#)

| [deleteFile \(page 338\)](#) | [setActiveTake \(page 1023\)](#)

continued...

System commands

deleteScriptPath (page 346)	setAltToSelect (page 1025)
exit (page 369)	setAutoSaveFile (page 1027)
formatTime (page 430)	setDir (page 1056)
frameToSmpte (page 432)	setErrorHandler (page 1063)
getActiveTake (page 441)	setExitCode (page 1065)
getAutoSaveFile (page 457)	 setFrameRate (page 1072)
getClipboardText (page 469)	setHotKey (page 1075)
getDirList (page 505)	setHotKeyFile (page 1079)
getFileExtension (page 526)	setLanguage (page 1085)
getFileList (page 528)	setLogEnabled (page 1103)
getFileLocation (page 531)	setLogFile (page 1104)
getFileName (page 533)	setMarkingMenuFile (page 1109)
getTitleFile (page 541)	setSavePath (page 1145)
getGlobalVarExists (page 558)	setSceneName (page 1149)
getLastScript (page 573)	setScriptPaths (page 1151)
getProfileInt (page 651)	setSelectionSetFile (page 1155)
getProfileString (page 653)	setTimeFormat (page 1182)
getSavePath (page 664)	smpteToFrame (page 1205)
getSceneName (page 668)	sortFloats (page 1231)
getScriptPaths (page 670)	sortInts (page 1233)

continued...

System commands

getSystemTime (page 695)	sortStrings (page 1235)
getTimecodeStandard (page 708)	system (page 1303)
help (page 739)	undo (page 1324)
listCommands (page 773)	undoConsolidated (page 1326)
listObjectTypes (page 775)	writeProfileInt (page 1360)
markingMenu (page 793)	writeProfileString (page 1363)

Testing commands

Use the Vicon Shogun Post script commands in this category for testing purposes.

- | [compareModules \(page 216\)](#)
- | [testPrint \(page 1310\)](#)

User window commands

Use the Vicon Shogun Post script commands in this category to manage a Shogun panel (docking window) or a user window.

User window commands

addDropListItem (page 105)	getNumBoxNum (page 610)
addListBoxItem (page 110)	getNumDropListItems (page 612)
addListViewItem (page 113)	getNumListBoxItems (page 616)
addTab (page 127)	getNumListViewColumns (page 618)
createCheckBox (page 239)	getNumListViewItems (page 620)
createColorPicker (page 242)	getRadioButtonCheck (page 657)
createDropList (page 247)	getTabItem (page 697)

continued...

User window commands

createForm (page 250)	getTabSelectedItem (page 699)
createGroupBox (page 253)	getTimeBoxTime (page 703)
createListBox (page 259)	getTopLevelForm (page 710)
createListView (page 262)	getWindowRect (page 723)
createNumBox (page 266)	getWindowVisibility (page 725)
createPushButton (page 271)	layoutForm (page 770)
createRadioButton (page 274)	selectDropListItem (page 982)
createStaticBox (page 297)	selectListBoxItem (page 989)
createTab (page 300)	selectListViewItem (page 991)
createTextBox (page 305)	selectTabItem (page 1007)
createTimeBox (page 308)	setCheckBoxCheck (page 1033)
createWindow (page 311)	setCheckBoxHandler (page 1035)
deleteAllDropListItems (page 324)	setColorPickerColor (page 1038)
deleteAllListBoxItems (page 326)	setColorPickerHandler (page 1040)
deleteAllListViewItems (page 328)	setControlAnchor (page 1045)
deleteAllTabItems (page 330)	setControlPos (page 1048)
deleteDropListItem (page 336)	setControlText (page 1051)
deleteListBoxItem (page 340)	setControlTip (page 1053)
deleteListViewItem (page 342)	setDropListHandler (page 1058)
deleteTabItem (page 352)	setFocus (page 1070)

continued...

User window commands

destroyWindow (page 358)	setListBoxHandler (page 1088)
dirChooser (page 360)	setListViewColumns (page 1091)
enableControl (page 365)	setListViewHandler (page 1094)
fileChooser (page 382)	setListViewItemCheck (page 1097)
findDropListItem (page 401)	setListViewItemText (page 1100)
findListBoxItem (page 407)	setNumBoxHandler (page 1111)
findTabItem (page 416)	setNumBoxNum (page 1114)
getCheckBoxCheck (page 465)	setNumBoxRange (page 1116)
getColorPickerColor (page 475)	setPushButtonHandler (page 1133)
getControlEnabled (page 489)	setRadioButtonCheck (page 1136)
getControlPos (page 491)	setRadioButtonHandler (page 1138)
getControlText (page 493)	setStaticBoxHandler (page 1160)
getControlVisibility (page 495)	setTabHandler (page 1167)
getDropListItem (page 510)	setTextBoxHandler (page 1170)
getDropListSellItem (page 513)	setTimeBoxHandler (page 1175)
getFocus (page 545)	setTimeBoxTime (page 1178)
getListBoxItem (page 578)	setWindowHandler (page 1188)
getListBoxSellItems (page 581)	showControl (page 1195)
getListViewItemCheck (page 583)	showWindow (page 1197)
getListViewItemText (page 586)	windowExists (page 1349)
getListViewSellItems (page 589)	

Python scripting with Vicon Shogun

For introductory information about using Python with Vicon Shogun Post, see the following topics:

- [Install the Shogun Post Python module \(page 1405\)](#)
- [Connect a Python client to Vicon Shogun Post \(page 1406\)](#)
- [Use Vicon Shogun Post SDK interfaces \(page 1407\)](#)
- [Switch the command line between HSL and Python \(page 1408\)](#)
- [Run Python scripts in Shogun Post \(page 1409\)](#)
- [Python / HSL interaction \(page 1410\)](#)
- [Run Python from ShogunPostCL \(page 1411\)](#)
- [Launch Python from the Start menu \(page 1412\)](#)

Install the Shogun Post Python module

When you install Vicon Shogun Post, it installs its own 64-bit version of Python 2.7.6 to the following default location:

C:\Program Files\Vicon\ShogunPost#.#\Python

The Python modules required by Shogun Post are included with the installation.

If you want to use a different 64-bit Python interpreter with Shogun Post, you can specify it in the Shogun Post **Preferences** dialog box, at the bottom of the **Directories** tab. If you change the interpreter, Shogun Post automatically installs the required modules to the site-packages folder. If necessary, you can also install the Shogun Post Python modules manually, using the installation script provided in the following default location:

C:\Program Files\Vicon\ShogunPost#.#\SDK\Install.py



Note

A 32-bit version of the SDK is available in *C:\Program Files\Vicon\ShogunPostX.X\SDK\Win32*.

When installed, the SDK is copied to *C:\Python27\Lib\site-packages* or a similar folder, depending on whether the 32- or 64-bit interpreter is used and which Python installation the SDK has been installed to.

Shogun Post Python modules

Shogun Post installs two Python modules:

- **ShogunPostSDK**. This folder contains the main Shogun Post SDK, and is the module that is most commonly used.
- **ViconShogunPost**. This module contains the Shogun Post implementation of a common Python SDK, shared by several Vicon products.

Connect a Python client to Vicon Shogun Post

When a Shogun Post Python client object is created it automatically tries to connect to a local instance of Shogun Post over TCP/IP via port 803.

```
import ViconShogunPostSDK
shogun = ViconShogunPostSDK.ViconShogunPost()
```

To direct the client to a different instance of Shogun Post, specify the appropriate parameters, or re-direct after creation using the `Connect()` method.

```
shogun = ViconShogunPostSDK.ViconShogunPost( "192.168.0.0", 804 )
```

You can change the port used by Shogun Post for communication in the **Preferences** dialog box, on the **Misc** tab. The default behavior of Shogun Post is to communicate via port 803, though if multiple instances of Shogun Post are opened on the same machine they will automatically choose different port numbers (searching incrementally upwards). You can specify a fixed port in the **Preferences** dialog box, or from the application command line.

```
ShogunPost.exe -ControlStreamPort 804
ShogunPostCL.exe -controlStreamPort 804
```



Tip

If you disconnect your Ethernet cable and disable wifi, when you enter a Python command, the following error may be displayed:

```
Host Application is not connected, unable to retrieve command list
```

This is because Python connects to Shogun Post over TCP/IP and if you are working entirely offline, Python and Shogun Post cannot connect.

To solve this issue, install the Microsoft Loopback Adapter. For instructions on how to do this, see *Adding the MS Loopback Adapter on Windows 7*, on <http://blogs.msdn.com>.

Use Vicon Shogun Post SDK interfaces

Two kinds of classes are defined in the *ViconShogunPostSDK* module. Classes like *Scene* or *Offline* define interfaces to distinct areas of Shogun Post's functionality. While you can create an instance of one of these interfaces, they can be more easily accessed directly from the Python client:

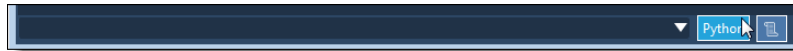
```
import ViconShogunPostSDK
shogun = ViconShogunPostSDK.ViconShogunPost()
shogun.Scene.Load( filename )
shogun.Offline.Reconstruct()
```

The other type of class in the SDK represents various content items in the Shogun Post scene, for example *Object* or *Attribute*. These can be instantiated freely and used to manipulate their corresponding objects in Shogun Post:

```
head = shogun.Scene.GetObject( 'Head' )
```

Switch the command line between HSL and Python

You can switch the command line in Shogun Post between HSL and Python.



The Python command line is essentially a mini script editor, not a direct interface to a Python interpreter, so variables will not persist from one call to the next. However, you don't need to create a client object when using the ShogunPost command line: default clients for the two Python modules are always available: `shogunPost` for the `ViconShogunPost` module, and `sdk` for the `ViconShogunPostSDK` module:

- `shogunPost.Command()` This is the `ViconShogunPost` module, which is the simple Vicon SDK.
- `sdk.Command()` This is the `ViconShogunPostSDK` module, which is the expanded Shogun Post-specific SDK.

Run Python scripts in Shogun Post

You can use Python scripts from anywhere in Shogun Post that currently accepts HSL scripts:

- General menu > Preferences dialog box > Directories tab.
- As a stage in a pipeline
- Attached to a shelf button



Note

When adding Python to a shelf button, a default shogunPost client is always available for use, as with the Shogun Post command line.

- Python command line (see [Switch the command line between HSL and Python \(page 1408\)](#))

Python scripts have an identifying icon in the Script Viewer. You can assign them to hot keys in the same way as HSL commands.

Python / HSL interaction

A command in HSL runs a Python command string, which works in the same way as the Shogun Post command line.

```
python "shogunPost.Command( )";
```

You can call a Python script from HSL in the same way as a native one:

```
MyPythonScript;
```

It can also be given arguments:

```
MyPythonScript( "Arg1", "Arg2");
```

Note

Returning values back from Python to HSL is not supported. You also cannot step into a Python script in the debugger.

You can call HSL from Python, using the following HSL command.

```
Result = shogunPost.HSL('hslString;')
```

The command returns a Python string containing the result of the HSL operation.

Run Python from ShogunPostCL

You can switch the ShogunPostCL command line between HSL and Python using the command `setLanguage`.

From HSL, the command is:

```
setLanguage "python";
```

From Python, the command is:

```
shogunPost.SetLanguage( "hsl" )
```

Launch Python from the Start menu

The Vicon program group in the Windows Start menu contains an item **ShogunPost Python** that contains a shortcut to the Python interpreter that is installed with Shogun Post.